VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Model vozidla a jeho pohyb ve virtuálním prostředí

Model of Vehicle and Its Movement in Virtual Environment

2018 Jan Hájovský

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Jan Hájovský**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Model vozidla a jeho pohyb ve virtuálním prostředí
Model of Vehicle and Its Movement in Virtual Environment

Jazyk vypracování: čeština

Zásady pro vypracování:

Cílem práce je vytvořit věrohodný model vozidla pohybujícího se ve virtuálním prostředí v rámci Unreal Engine (UE). Mechanický model vozidla musí obsahovat dvojité lichoběžníkové zavěšení náprav s možností natáčení předních kol. Po vizuální stránce se musí jednat o propracovaný model vozidla typu SUV s kompletním otexturováním a s vhodnými materiály. Při vypracovávání zadání postupujte podle následujících bodů:

1. Prostudujte možnosti vytváření realistických modelů vozidel v prostředí UE.
2. Nalezněte vhodný 3D model vozidla SUV [3] a upravte ho do podoby, která je efektivně použitelná v prostředí UE.
3. Výsledky bodů 1 a 2 propojte do podoby vozidla pohybujícího se v komplexním terénu.
4. Vytvořte sadu materiálů vhodných pro model vozidla a terén. Vozidlo bude zanechávat stopy po pneumatikách.
5. Umožněte pohyb vozidla pomocí skriptování akcí jako plynulé přidávání plynu, otáčení volantu, brždění apod.
6. Dosažené výsledky demonstrujte formou hratelného dema.
7. Použité postupy pečlivě zdokumentujte v textové části práce.

Seznam doporučené odborné literatury:

[1] Tavakkoli, A. Game Development and Simulation with Unreal Technology. AK Peters, ISBN 978-1498706247, 741 p., 2015.
[2] Sewell, B. Blueprints Visual Scripting for Unreal Engine. Packt Publishing, ISBN 978-1785286018, 188 p., 2015.
[3] Poitschke, T., et al. A Multifunctional VR-Simulator Platform for the Evaluation of Automotive User Interfaces. In Proc. 12th International HCI Conference, p. 1120-1129., 2007.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Tomáš Fabián, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 30.04.2018

doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 26. dubna 2018 ..... Hájovský Jan .....

Souhlasím se zveřejněním této diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě 26. dubna 2018 ..Hájovský Jan.....

Rád bych na tomto místě poděkoval svému vedoucímu Ing. Tomáši Fabiánovi, Ph.D. za příkladné vedení a užitečné rady během zpracovávání této diplomové práce. Dále bych chtěl poděkovat všem rodinným příslušníkům a osobám mně blízkým, kteří mě při mé činnosti soustavně podporovali.

Abstrakt

V této diplomové práci se zabývám věrohodnou simulací jízdy vozidla v terénu. Věrohodnosti je dosaženo nejen použitím nezávislého lichoběžníkové zavěšení vozidla, ale také odpovídajícím zanecháváním stop pneumatik či dalšími efekty (prach, kouř). Při řešení jsem použil 3ds max na úpravu volně dostupného 3D modelu vozidla, rozšíření o jednotlivé objekty tohoto typu nápravy a jejich propojení. Výsledek je importován do prostředí Unreal Engine, ve kterém je vozidlo rozpohybováno. Věrohodnosti také přidává vhodné použití efektů v závislosti na povrchu, po kterém se vozidlo pohybuje. Výstupem je hratelné demo s možností automatické jízdy vozidla po předem vytyčené trase, což usnadňuje demonstraci pohybu jednotlivých komponent nápravy.

Klíčová slova: simulace, náprava, lichoběžníkové zavěšení, 3D modelování, 3ds max, Unreal Engine, C++

Abstract

In this diploma thesis I deal with a credible simulation of vehicle driving in terrain. Credibility is achieved not only by using an independent double wishbone suspension of the vehicle but also by leaving appropriate the tire tracks or other effects (dust, smoke). In the solution, I used 3ds max to modify the freely available 3D model of the vehicle, extending the individual objects of this suspension type and their interconnection. The result is imported into the Unreal Engine in which the vehicle is turned into motion. Credibility also adds the appropriate application of effects depending on the surface the vehicle is moving. The output is a playable demo with the option of automatic drive of the vehicle on a predefined route, making it easier to demonstrate the movement of each component of the suspension.

Key Words: simulation, axle, double wishbone suspension, 3D modeling, 3ds max, Unreal Engine, C++

Obsah

Seznam použitých zkratek a symbolů **10**

Seznam obrázků **11**

Seznam tabulek **12**

Seznam výpisů zdrojového kódu **13**

1 Úvod **14**

2 Mechanika zavěšení kol **16**

3 Unreal Engine **18**

4 Model vozidla **19**
4.1 Získání modelu 19
4.2 Technické údaje vozidla 19
4.3 Opravy získaného modelu 19

5 Vlastní modelování v 3ds max **22**
5.1 Pozicování pivotů v 3ds max 24
5.2 Vzájemné propojení součástí modelu 24
5.2.1 Rigování modelu 26
5.2.2 Specifická příprava pro UE 28
5.2.3 Export modelu 28

6 Vytvoření projektu **30**

7 Tvorba terénu **31**

8 Vlastní C++ třídy **34**
8.1 Vytvoření efektů 35

9 Sdílený blueprint pro auta **36**
9.1 Změna kamery 36
9.2 Skrytí částí modelu vozidla 37
9.3 Nastavení tempomatu 38
9.4 Udržování natočení kol 39
9.5 Zanechávání stop pneumatik 40
9.5.1 Vytvoření materiálu 40

9.5.2 Generování stop 41

10 Land Rover **45**

11 Nastavení Level blueprint **49**

12 Závěr **50**

Reference **52**

Přílohy **53**

A Snímky obrazovky ze simulace **54**

B MaxScript funkce **59**

C Hlavičkový soubor třídy NastaveniEfektu **60**

D Zdrojový soubor třídy NastaveniEfektu **62**

E Hlavičkový soubor třídy HAJ0094_efekty **63**

F Zdrojový soubor třídy HAJ0094_efekty **64**

Seznam použitých zkratek a symbolů

SUV	–	Sport Utility Vehicle
UE	–	Unreal Engine
FPS	–	First-person Shooter
RPG	–	Role-playing Game
HSE	–	High Specification Edition
FBX	–	Filmbox

Seznam obrázků

1 Schéma lichoběžníkové nápravy se zvýrazněným lichoběžníkem . . . 16
2 Znázornění lichoběžníkové zavěšení . . . 17
3 Přední lichoběžníková náprava automobil Škoda 130 . . . 17
4 Ukázkový graf . . . 18
5 Pojmenování objektů . . . 20
6 Špatná orientace normál . . . 20
7 Model vozidla Land Rover po nutných opravách ve 3ds max . . . 21
8 Model podvozku . . . 22
9 Pivot spodního ramene . . . 23
10 Pivot těhlice . . . 23
11 Váhy kostí pro tlumič, vlevo pro horní kost, vpravo pro dolní . . . 27
12 Váhy kosti b_pl_tehlice_stred pro poloosu . . . 27
13 Simulovaný pohyb kola . . . 28
14 Reálný pohyb kola . . . 28
15 Kompletní hierarchie kostí pro levé přední kolo . . . 29
16 Zvrásnění terénu . . . 31
17 Vytvoření záznamové vrstvy pro typ povrchu . . . 32
18 Nastavení vlastních jmen pro typy povrchů . . . 33
19 Terén s překážkami a doplňkovými objekty . . . 33
20 Moduly částicových systémů pro asfalt, šotolinu a trávu . . . 35
21 Metoda ZmenKameru . . . 37
22 Inicializace standardní kamery a kontrola stisknutých tlačítek . . . 37
23 Metoda ZvolRychlost . . . 38
24 Metoda ZmenRychlostNa . . . 38
25 Plynulé natáčení kol při stisku tlačítka . . . 39
26 Textura zanechávaných stop . . . 40
27 Materiál pro zanechávané stopy . . . 40
28 Podmínky kreslení stop . . . 43
29 Aktualizace informací o posledních generovaných stopách . . . 43
30 Generování běžných stop vozidla . . . 44
31 Kompletní animace pro přední levé kolo . . . 47
32 Vyvolání události předání údajů v blueprintu AutoSEfekty . . . 48
33 Obsloužení události předání údajů v animačním blueprintu . . . 48
34 Úprava vstupů a provedení transformací v animačním blueprintu . . . 48
35 Restartování simulace . . . 49

Seznam tabulek

1 Potřebné proměnné pro správné generování stop 41
2 Tabulka animací pro přední levé kolo 46

Seznam výpisů zdrojového kódu

1 Posun kostí kol a brzd na pivot kola 25
2 Vlastní MaxScript funkce 59
3 NastaveniEfektu.h 60
4 NastaveniEfektu.cpp 62
5 HAJ0094_efekty.h 63
6 HAJ0094_efekty.cpp 64

1 Úvod

Cílem práce je nastudovat možnosti simulace jízdy vozidla typu SUV v terénu a takovou simulaci následně provést, zároveň by měl být k simulaci použit Unreal Engine.

Nejprve je však nutné seznámit se se zavěšením kol, jenž je stěžejní část věrohodné simulace. Zavěšení kol je mnoho typů, v práci ale bude použito nezávislé lichoběžníkové zavěšení, velice rozšířené právě u vozidel typu SUV.

Modelování vozidla od základu není cílem práce, tak je možno použít již nějaký existující model. Ten bude nutné opravit, pokud jsou v něm zásadní nedostatky. Model vozidla bude také nutné rozšířit o jednotlivé části náprav. Takto vymodelované součásti bude nutné propojit systémem kostí, které umožní další manipulaci s objekty. Rovněž bude nutné zjistit, jak tato propojení vhodně provést, aby byl model co nejsnáze použitelný v dalších krocích, respektive po importu do Unreal Engine.

V Unreal Engine bude nutné nastudovat, jak připravený model efektivně použít. V této části také dojde ke zprovoznění vozidla a rozpohybování jednotlivých částí nápravy. Použití různých pohledů by následnému uživateli mělo ulehčit pochopení principu použitého typu zavěšení. Dále auto po sobě musí zanechávat stopy pneumatik a rovněž by byly vhodné i odpovídající efekty (například kouř od pneumatik). Nedílnou součástí této části bude také vytvoření či přepoužití materiálů a textur pro docílení co nejlepšího vzhledu.

V neposlední řadě bude také nutno připravit terén, po kterém se bude vozidlo pohybovat. Jako každá testovací trať i tato bude mít více povrchů, přičemž jak pohyb vozidla, tak i efekty by se měly mezi jednotlivými povrchy lišit podle očekávání z reálného světa. Pro snažší demonstraci funkce zavěšení by bylo vhodné mít možnost automatické jízdy vozidla po připravené trase.

Cílem práce je vytvořit hratelné demo a na něm ukázat nejen chování auta jako celku při průjezdu terénem, ale také reakce jednotlivých součástí zavěšení kol na překonávané nerovnosti. Pohyb těchto součástí by měl odpovídat pohybu u reálného auta. Zároveň by simulace měla být z grafického hlediska zajímavá a líbivá, aby v člověku vzbudila zvědavost a chuť poznávat.

Vývoj simulací obecně se pohybuje rychlým tempem a dnešní možnosti jsou naprosto odlišné od těch před pěti či deseti lety. První automobilové hry začaly vznikat kolem roku 1974, přičemž jako průkopník tohoto žánru je označována hra Gran Trak 10 od Atari. Jednalo se o, z dnešního pohledu, velice jednoduchou hru s pohledem z vrchu, jejíž náplní je projíždět trať do určitého časového limitu. Vývoj pokračoval dále, přičemž další známější hrou je Out Run z roku 1986 od firmy Sega. Ten už nabízí pohled na jedoucí auto zezadu, což je používané dodnes a nejinak tomu bude i v mé práci. V roce 1988 představila firma Accolade hru z názvem Grand Prix Circuit, která pro změnu nabídla ovládání ze sedačky řidiče. V devadesátých letech vznikla další známá hra Stunts, která nabídla možnost přidávání překážek na trať, například různé rampy, mosty a jiné. Pokud jste dosud neznali žádnou zmíněnou hru, jistě to napraví rok 1994 s představením první hry ze série The Need For Speed. V roce 1997 došlo k vydání první verze

TORCS (The Open Racing Car Simulator), která je stále vyvíjena s licencí GNU GPL. V roce 2004 spatřila světlo světa hra Richard Burns Rally, které v sobě mělo velice povedený fyzikální model. O populárnosti této hry také svědčí také aktivní komunita, díky které stále vznikají nové doplňky a rozšíření. Neméně známá je také hra Flatout ze stejného roku. V následujících letech vzniklo množství skvělých her, namátkou bych zmínil třeba Test Drive Unlimited z roku 2006, Gran Turismo 5 z roku 2010, Forza Motorsport 6 z roku 2015 či Dirt Rally z roku 2017 [1].

V poslední době je hodně zajímavé sledovat vývoj titulu BeamNG.drive, který se zaměřuje na realistickou simulaci pohybu vozidla. Mezi jeho přednosti patří simulace jednotlivých komponent vozidla, čímž je dosaženo realistického chování celého auta. To se projevuje zejména při simulacích nárazů, kdy se jednotlivé části vozidla deformují očekávaným způsobem či dokonce zdeformované odpojují. Existuje také mnoho porovnání simulovaného nárazu a reálně provedeného crash testu, přičemž rozdíly jsou často velice malé.

Zároveň se vyvíjí také herní enginy, ve kterých je možno vlastní simulaci vytvořit. Nejznámější v době psaní práce jsou Unreal Engine a Unity, zajímavý se mi však jeví třeba také ProjectChrono či V-REP. Unreal Engine je multiplatformní engine, vyvíjený firmou Epic Games. Primární určení bylo pro FPS hry (střílečky z vlastního pohledu), rozšířil se však i do dalších žánrů jako RPG či adventur. Více informací o Unreal Enginu je v kapitole 3 [2]. Multiplatformní engine pro vývoj her je i Unity, vyvíjený firmou Unity Technologies. Unity podporuje také mobilní platformy, čehož využívá například hra Pokémon Go. Samotný engine je naprogramovaný v C++ a uživatel jej může rozšiřovat zejména C# skripty [3]. Třetí zmíněný, ProjectChrono je multiplatformní open source systém založený na knihovně Chrono. Ta poskytuje primárně C++ rozhraní, existují však rozšíření pro Python (Chrono::PyEngine) či dokonce SolidWorks (Chrono::SolidWorks). Knihovna disponuje také rozšířením Chrono::Vehicle, které je určeno k simulaci kolových a pásových vozidel a spojuje simulaci jednotlivých prvků vozidla (pohonná jednotka, pneumatiky, zavěšení kol) v jeden celek. Zajímavostí je také fakt, že armáda Spojených Států Amerických investovala do projektu 1,8 miliónu dolarů, aby mohl vývoj pokračovat stále pod svobodnou licencí [4]. Poslední zmíněný V-REP (Virtual Robot Experimentation Platform) je obecný simulátor, určený prvotně pro simulaci robotů. Je založen na tom, že každý objekt či model může být individuálně řízen a to hned několika různými způsoby, od předem zvoleného skriptu po kopii chování reálného robota. Používá se například k vývoji algoritmů, vzdálený monitoring, simulaci automatizace továren a jiné [5].

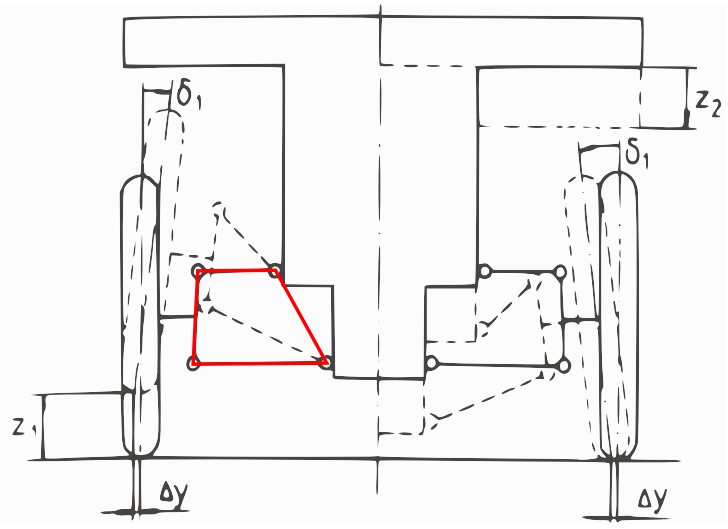
2 Mechanika zavěšení kol

Zavěšení kol je jedna z klíčových součástí každého vozidla. Slouží nejen k uchycení kol, kde umožňuje jejich pohyb či natáčení, ale také k uchycení tlumící soustavy. Tyto úlohy přímo ovlivňují jízdní vlastnosti vozidla a v žádném případě nesmí být opomenuty při věrohodné simulaci pohybu vozidla.

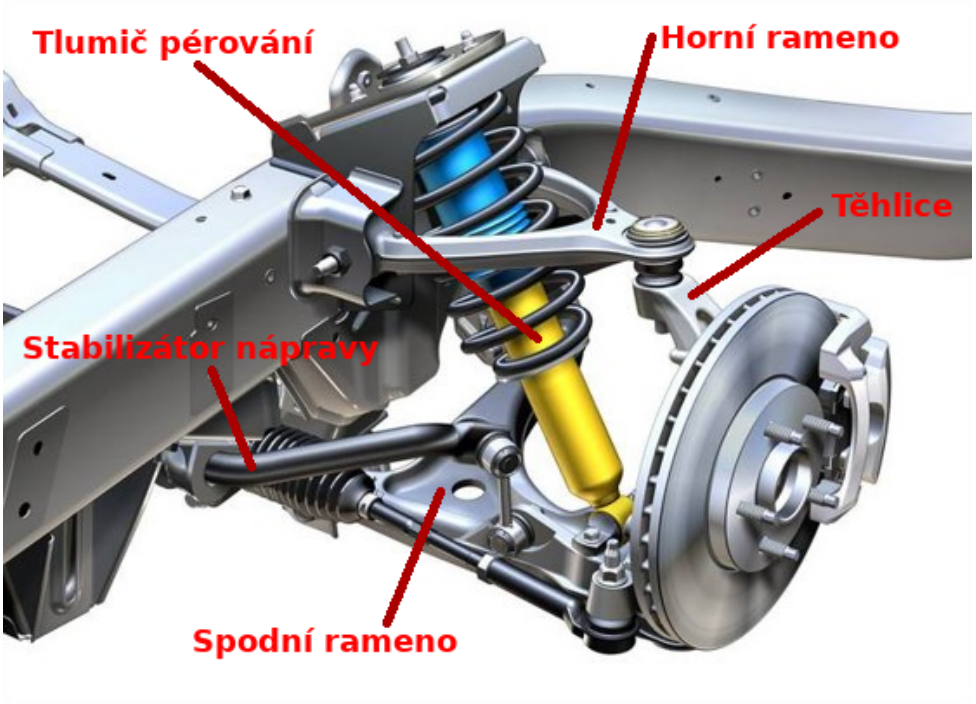
Zavěšení kol se dělí na závislé a nezávislé. Závislé zavěšení (jinak také tuhá náprava) má kola umístěna na společném nosníku, což znamená, že pohyb jednoho kola vyvolá pohyb druhého kola. Tuhou nápravu mohou čeští motoristé znát například ze zadních náprav vozů Škoda Felicia, Fabia či Octavia. Oproti tomu při nezávislém zavěšení má každé kolo své vlastní uchycení, čímž pádem reakce jednoho kola neovlivňuje druhé kolo. Tento systém zavěšení je běžně použit u řídících náprav či všech náprav (nejen) vozů typu SUV.

Jeden z typů nezávislého zavěšení je lichoběžníková náprava, která se skládá ze dvou nad sebou umístěných příčných ramen obvykle trojúhelníkového tvaru. Ramena jsou na širší straně pevně přichycena ke karoserii, na druhé straně se nachází čepy pro uchycení těhlice (hlava ložiska čepu kola). Na těhlici je pak dále uchycen brzdový kotouč, třmen i samotné kolo. Jelikož horní rameno bývá trochu kratší, vytvářejí ramena spolu s těhlicí při pohledu zepředu lichoběžník, což dalo i název pro tento typ zavěšení. Schéma zavěšení i zvýrazněný lichoběžník je možno vidět na obrázku 1 [6]. Použitím vhodné délky obou ramen lze dosáhnout velice zajímavých jízdních vlastností, proto je tato náprava vhodná jako řídící, může však být použita i jako hnací [7].

Zajímavostí je také například to, že lichoběžníkovou nápravu měla i plejáda vozů značky Škoda, mimo jiné například Škoda 1000 MB, Škoda 105 nebo Škoda 120.



Obrázek 1: Schéma lichoběžníkové nápravy se zvýrazněným lichoběžníkem, převzato z [6] a upraveno



Obrázek 2: Znázornění lichoběžníkové zavěšení, převzato z [8] a upraveno

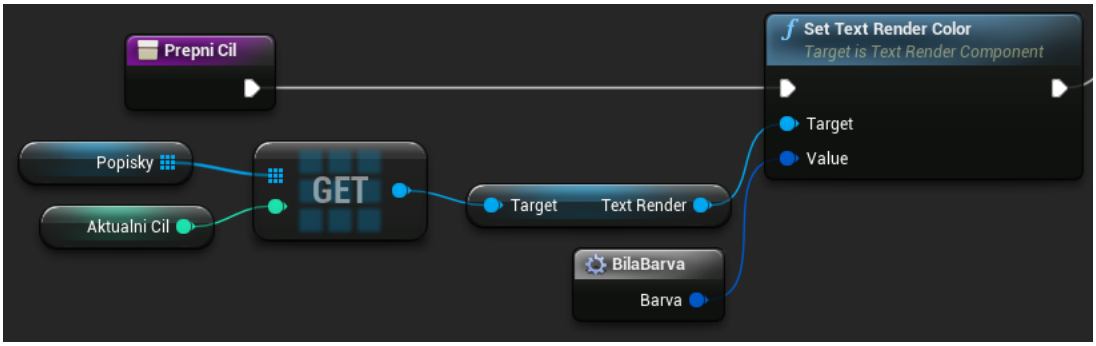
Obrázek 3: Přední lichoběžníková náprava automobil Škoda 130, převzato z [9]

3 Unreal Engine

V úvodu jsem již naznačil, jaké možnosti z pohledu herních enginů a knihoven má zájemce k dispozici. Já jsem se rozhodl použít Unreal Engine, jelikož je v současné době hodně rozšířený, aktivně vyvíjený a s širokou komunitou lidí okolo.

Samotný engine je vyvíjený firmou Epic Games [2]. Poskytuje kompletní sadu pro rychlý vývoj her a jeho vývoj započal již v roce 1998. V době psaní práce je aktuální verze 4, přesněji verze 4.18.3. UE podporuje mnoho platforem, jak počítačové Windows, Linux, OS X, tak mobilní jako Android či iOS, tak i herní konzoly Xbox či PlayStation. Zároveň je plně připraven na virtuální realitu podporou Oculus, Vive či Samsung Gear VR. Mezi nejznámější hry, vyvinuté právě v tomto enginu, patří Unreal Tournament, Bioshock, XCOM: Enemy Unknown či Mass Effect 2.

V UE 4 byl představen Blueprints Visual Scripting systém, zkráceně blueprinty. Jedná se o způsob vývoje umožňující kompletní vývoj hry spojováním odpovídajících akcí ve formě grafu. Připravených akcí je obrovské množství, které by jinak musel programátor řešit sám. Od základního nastavení proměnných přes volání funkční či vyvolávání událostí. Samozřejmostí je možnost vytvoření blueprintu uživatelem, což možnosti uživatele dále rozšiřuje. Jak takový graf vypadá můžete vidět na obrázku 4 [10, 11]. Pokud však blueprinty nějakým způsobem nestačí, je možno implementovat požadovanou funkcionalitu pomocí C++, kde už jsou možnosti takřka neomezené [12, 13].



Obrázek 4: Ukázkový graf

4 Model vozidla

Abych měl simulaci na čem provést, potřebuji mít k dispozici nějaký model auta. Jeho získání a nutné opravy či úpravy popíšu v této kapitole. Zároveň k těmto pracím potřebuji použít nějaký modelovací nástroj, přičemž já jsem zvolil 3ds max od firmy Autodesk.

4.1 Získání modelu

Než jsem mohl začít s prací, musel jsem najít vhodný model. Jelikož cílem nebylo vymodelovat celý model od nuly, vyhledával jsem modely vozidel typu SUV k volnému použití. Cena takovýchto modelů se různí, jsou modely zadarmo i modely s cenou v řádu jednotek tisíců Kč a výše. Já jsem vybíral pouze modely zdarma, aby mohl případně kdokoli mou práci bez nutnosti dalších finančních investicí reprodukovat. Zvolil jsem model vozidla Range Rover dostupný na webu cgtrader.com [14].

4.2 Technické údaje vozidla

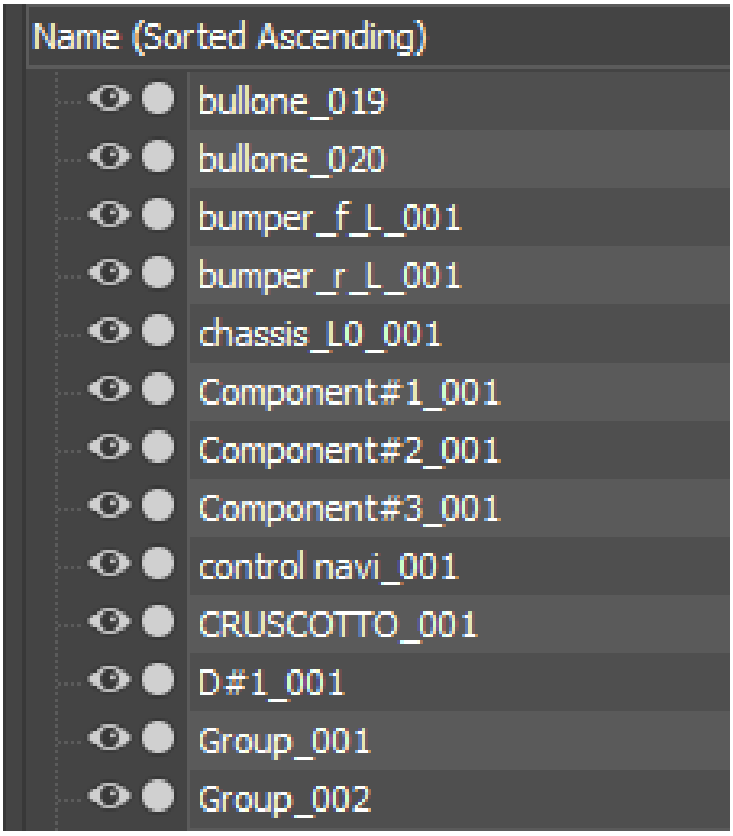
Jedná se o vrcholnou verzi Range Rover Sport ve speciální edici HSE. Vozidlo bylo v této podobě vyráběno v letech 2004 až 2013, kdy byla představena druhá generace. Edice HSE byla od roku 2010 poháněna přeplňovaným benzínovým vidlicovým osmiválcem o objemu 5.0 litru. Maximální výkon tohoto motoru činí 510 koní (375 kW) při 6000 otáčkách a křivka točivého momentu dosahuje maxima 625 Nm při 2500 otáčkách [15]. Motor doplňuje šestistupňová automatická převodovka ZF 6HP28 [21]. Zavěšení kol je nezávislé lichoběžníkové a tlumení nerovností zajišťuje vzduchový podvozek [15].

4.3 Opravy získaného modelu

Kvalita modelů zdarma však zpravidla není vždy optimální, což se ukázalo hned vzápětí i na mnou zvoleném modelu. Model vypadá po vzhledové stránce pěkně, při manipulaci s jeho strukturou jsem však objevil několik, místy zásadních, problémů. Prvním viditelným bylo pojmenování jednotlivých částí objektu, viz obrázek 5. Jména objektů nedávala smysl a i objekty s podobnými názvy spolu vůbec nesouvisely. Pro lepší budoucí manipulaci s objektem bylo nutné tato jména opravit a spolu související objekty vhodně sloučit do skupin (páté dveře se skládají ze samotného plechového dílu, okna, nápisů, spz apod.)

Poté jsem model zkusil nahrát do prostředí Unreal Engine [2]. Zde se projevil, tentokrát už vážnější, problém s normálami jednotlivých objektů, jenž byly v některých objektech či jejich částech špatně natočeny, viz obrázek 6. Při manuálních opravách jsem narazil na množství mnohostěnů, jenž nesplňovaly Eulerovu větu, pomocné nástroje ke sjednocení normál tedy nefungovaly korektně či vůbec. Bylo tedy nutné provádět opravy manuálně.

Pak jsem se věnoval přípravě materiálů. Materiály na objektech byly podobně zmatečné, jako dříve pojmenování objektů. Materiály byly mnohdy duplikovány a ve výsledku měl model



Obrázek 5: Pojmenování objektů

Obrázek 6: Špatná orientace normál

po načtení v Unreal Engine přesně 199 materiálů. Připravil jsem si tedy sadu několika materiálů, které jsem postupně aplikoval na odpovídající polygony jednotlivých objektů. Nejedná se o těžkou práci, je však hodně zdlouhavá a člověk v ní snadno ztratí přehled, jelikož je materiály nutné nastavit pro jednotlivé polygony. Samotné vlastnosti materiálů jsem však příliš neřešil, při exportu jsou přeneseny pouze základní vlastnosti a zároveň Unreal Engine poskytuje lepší nástroje pro tvorbu realistických materiálů. Na výsledek se můžete podívat na obrázku 7.

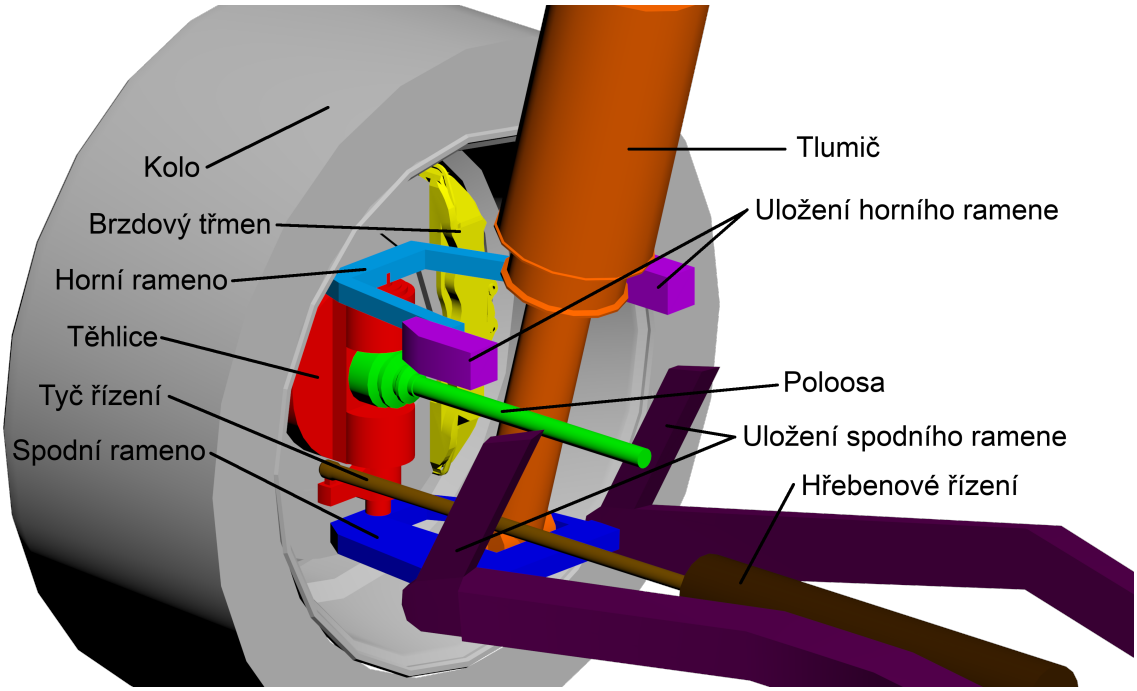
Obrázek 7: Model vozidla Land Rover po nutných opravách ve 3ds max

5 Vlastní modelování v 3ds max

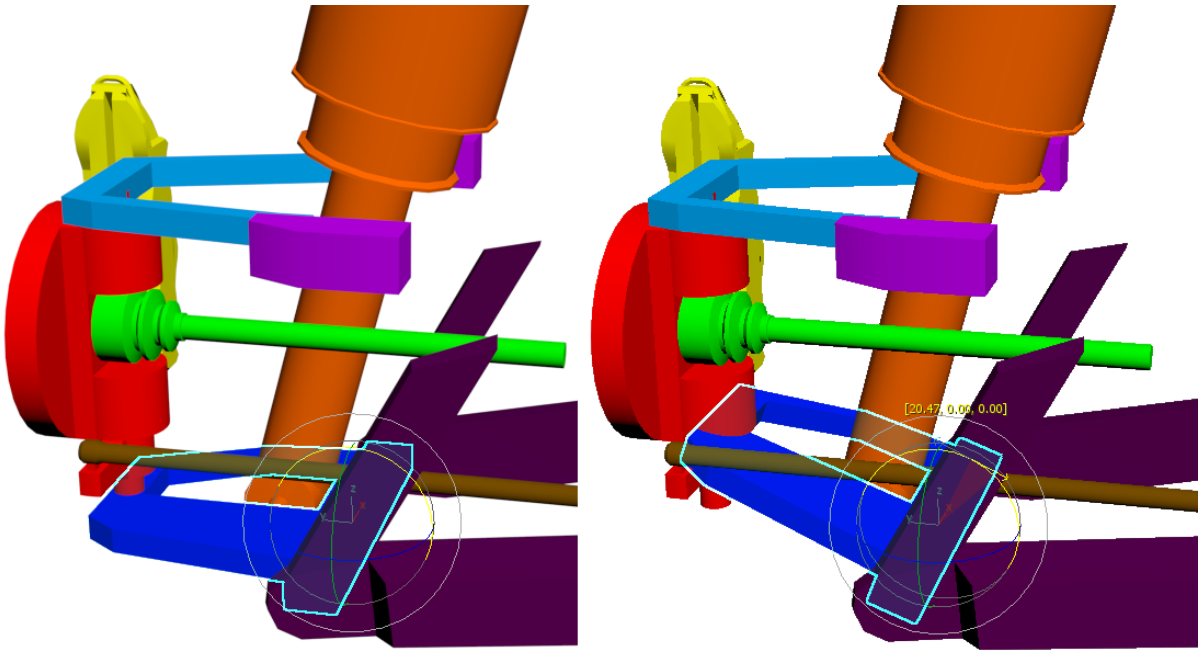
Jedním z cílů této práce je simulace právě nezávislého zavěšení náprav. Nápravy při běžném použití modelu vidět a většina zdarma dostupných modelů je vymodelovány vůbec nemá. A když má, tak velice zjednodušeně a tedy nevhodně pro tuto práci. V mnou získaném modelu vůbec nebyly, musel jsem je tedy dodělat sám.

Začal jsem nejprve vymodelováním uložení náprav, což jsou pevná místa na karoserii, kde jsou uchycena ramena. Abych dodržel lichoběžníkový typ zavěšení, uložení pro horní rameno jsem umístil blíže ke kraji vozidla. Mezi opačné konce ramen jsem vymodeloval těhlici. V polovině vzdálenosti konců ramen jsem vytvořil vykrojení pro připojení poloosy a rovněž jsem ji vymodeloval. Zároveň pro přední nápravu bylo nutné na těhlici přidat čep řízení, pro nějž jsem následně vytvořil i tyč řízení. Poslední stěžejní část je tlumič. Ten je v horní části pevně uchycen v karoserii, spodní část ale patří připojit ke spodnímu rameni. Při modelování je dobré myslet na to, aby se jednotlivé součásti nepřekrývaly, ani když se bude náprava pohybovat. Všechny potřebné objekty je možno vidět na obrázku 8.

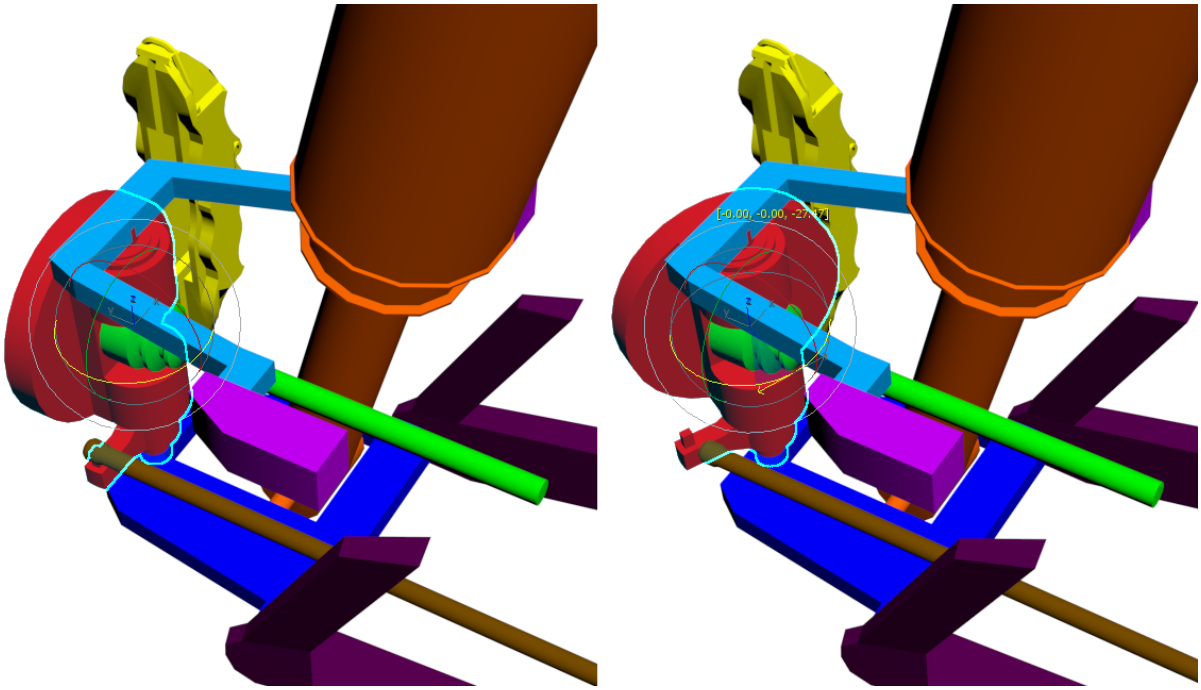
Samotné vymodelování jednotlivých částí je z mého pohledu ta lehčí část problému. Náročnější je nastavení správných pivotů (os otáčení) všech objektů nápravy, které je kritické pro reálnou vizualizaci funkce. Ukázku správné pozice pivotů některých objektů je možno vidět na obrázcích 9 a 10. Pivoty kola a brzového třmenu musí být stejné jako pivot těhlice, jelikož jsou na ní připevněny. Pokud budou pivoty umístěné jinak, je nutno upravit i další kroky této práce.



Obrázek 8: Model podvozku



Obrázek 9: Pivot spodního ramene



Obrázek 10: Pivot těhlice

5.1 Pozicování pivotů v 3ds max

V 3ds max jde pivot graficky libovolně posouvat [16], není však možné jeho pozici přesně pozicovat, tím mám na mysli přichytit jej k určitému bodu či těžišti několika bodů. Tento problém se však dá vyřešit za použití MaxScriptu. Vytvořil jsem si pomocné funkce, které jsou schopny mi z aktuálně vybraných vertexů `plv` nebo polygonů `plf` objektu vypočítat těžiště pozic (plv - PoLyVertices, plf - PoLyFaces). Funkce jsou přiloženy v příloze B. Je nutné však mít objekty převedeny do režimu `Editable Poly`, jelikož jiné typy, například `Editable Mesh`, mají jiná rozhraní a tedy nabízené funkce. Takovou funkci je pak možno z `MaxScript Listeneru` zavolat `plf $pl_horni_rameno`. Označení za znakem dolaru je název objektu, z jehož vybraných bodů chceme spočítat průsečík. Dále jsem zjistil, že každý objekt má zároveň i atribut `pivot`, který můžu rovněž nastavovat, vylepšil jsem tedy svůj skript o funkce `plvs` a `plfs`, jež značí `PoLyVertices Set`, respektive `PoLyFaces Set`.

5.2 Vzájemné propojení součástí modelu

Když jsou osy otáčení řádně nastaveny, je nutné jednotlivé součásti také propojit. Tím mám na mysli provázání akcí objektů, aby například pohyb kola vyvolal pohyb těhlice, ta pohne ramenem, to následně vyvolá stlačení tlumiče apod. K tomu slouží kosti (označované také jako Armature či Bones) a můžeme je nalézt v každém moderním a běžně rozšířeném modelovacím nástroji. Pro přehlednost je dobré pojmenování kostí odlišit od dalších objektů, k čemuž jsem já zvolil použití předpony `b_`.

Tvorba kostí

Základní kost pro celé auto jsem pojmenoval `b_root` a v modelovacím nástroji se nachází v bodě `[0,0,0]`. Jakékoliv další kosti budou z pohledu hierarchie potomci této kosti, přičemž potomci dědí provedené transformace právě nadřazené kosti. První vnořenou kost jsem pojmenoval `b_karoserie`, která je na stejném místě a slouží pro pevné díly na autě jako je karoserie, sedačky či skla. Tyto díly by mohly být připojeny i na hlavní kost `b_root`, pro lepší ukázku funkce zavěšení jsem se ale rozhodl přidat možnost schování těchto objektů. Aby se při tomto zobrazení ramena opticky nevznášela v prostoru, objekty pro uložení ramen jsou připojeny k hlavní kosti a jsou tedy zobrazeny vždy. Další kosti budou kosti pro kola a brzdové třmeny. Oddělené kosti pro kolo a třmen musí být z důvodu toho, že brzdový třmen se bude naklánět do stran stejně jako kolo, nebude však za jízdy rotovat.

Rovněž i při pozicování kostí platí fakt, že aby byl výsledek pěkný, je nutné pozicovat přesně. Proto jsem opět využil MaxScript. Každý objekt má totiž vlastnost `pos`, kde je definována poloha v souřadném systému. A jelikož kost je objekt, nic tedy nebrání nastavit tuto jeho vlastnost na pivot objektu, který kost bude ovládat. Pokud byly tedy pivoty správně nastaveny, je posun kostí pro kolo a brzdový třmen za pomocí `MaxScriptu` blesková záležitost, což znázorňuje výpis 1.

```
$b_pl_kolo.pos = $pl_tehlice.pivot
$b_pl_brzda.pos = $pl_tehlice.pivot
$b_pp_kolo.pos = $pp_tehlice.pivot
$b_pp_brzda.pos = $pp_tehlice.pivot
$b_zl_kolo.pos = $zl_tehlice.pivot
$b_zl brzda.pos = $zl_tehlice.pivot
$b_zp_kolo.pos = $zp_tehlice.pivot
$b_zp_brzda.pos = $zp_tehlice.pivot
```

Výpis 1: Posun kostí kol a brzd na pivot kola

U některých kostí jsem pro větší přehlednost přidal do názvu kosti i její případný cíl. Tak vznikla `b_pl_dolni_rameno_tehlice`, která začíná v pivotu spodního ramene a končí v pivotu těhlice. Tato kost bude mít za cíl sledování pohybu simulovaného kola z Unreal Engine. Pokud se kolo pohne nahoru či dolů, pohne se odpovídajícím dílem i rameno. Tato kost je také nadřazena dvěma dalším. První z nich je `b_pl_dolni_cep`, která se nachází v místech spojení těhlice a spodního ramene. V reálném světě se právě toto spojení označuje jako spodní či dolní čep. Druhá vnořená kost `b_pl_tlumic_uchyceni` je spojena s tlumičem, který je uchycen na vhodném místě spodního ramene.

Těhlice je sice přichycena na spodním rameni, jelikož ale nechci kopírovat její rotaci, je nutné, aby měla svou kost, tak jsem vytvořil `b_pl_tehlice_dolni_cep`. Tato kost bude sloužit k nastavení pozice a rotace těhlice. Jednou ze čtyř klíčových bodů na těhlici (a tedy vnořenou kostí) je `b_pl_horni_cep`. To je místo, kde bude přichycen konec horního rameno. Druhá kost je `b_pl_tehlice_stred`, na kterou bude při animaci přichyceno kolo, brzdový třmen a poloosa. Třetí kost `b_pl_kolo_efekty`, která se nachází nejlépe na nejnižším bodě kola, použiji pro vytváření částicových systémů jako například prach. Název kosti by mohl svádět k přesunutí pod kost `b_pl_kolo`, nicméně umístěním pod těhlici ušetřím několik operací v dalších krocích, jelikož by tato kost kopírovala rotaci kola. Poslední kost na těhlici bude pouze pro přední nápravu, jedná se o `b_pl_cep_rizeni`. Je to místo, kde končí tyč řízení, vycházející z hřebenového řízení.

Horní rameno už není nijak obtížné, pro něj stačí jedna kost `b_pl_horni_rameno_tehlice`, která začíná v pivotu horního ramene a směřuje na horní čep těhlice. V těhlici končí poloosa, která vychází z diferenciálu, proto na tomto konci také musí existovat kost `b_pl_poloosa_diferencial`. Ta bude potomek hlavní kosti `b_root`, jelikož diferenciál zůstává vždy na stejném místě. Stejně tak je nutné přidat kost na výstup z hřebenového řízení s posilovačem (servem) řízení, kost jsem pojmenoval příznačně `b_pl_servorizeni`.

Tlumič je trochu specifický a je potřeba mít pro něj kosti dvě, v horním a dolním uchycení tlumiče. Budou potřeba proto, abych mohl správně nastavit stlačování a roztahování tlumiče a natočení tlumiče při těchto akcích. Kosti tlumičů je nutné mít natočeny ve směru tlumiče, což v mém případě znamenalo rotaci podle osy X o 15°. V animaci totiž bude horní kost sledovat dolní a naopak. Pokud by kost nebyla natočena zde, při spuštění animaci by došlo k natočení

této kosti, což by vyvovalo další, nechtěné, natočení tlumiče.

Pro pěknou simulaci i v rámci interiéru jsem přidal jednu kost pro volant `b_volant` a 2 další pro ukazatele budíků, otáčkoměr `b_otackomer` a rychloměr `b_rychlomer`. Zde je, podobně jako u tlumičů, nutné natočení ve vhodné ose, aby stačilo pro natočení vykonat rotaci dle jedné osy. V mém případě jsem volantovou kost natočil o 25° podle osy Y a kosti budíků o 15° rovněž podle osy Y.

5.2.1 Rigování modelu

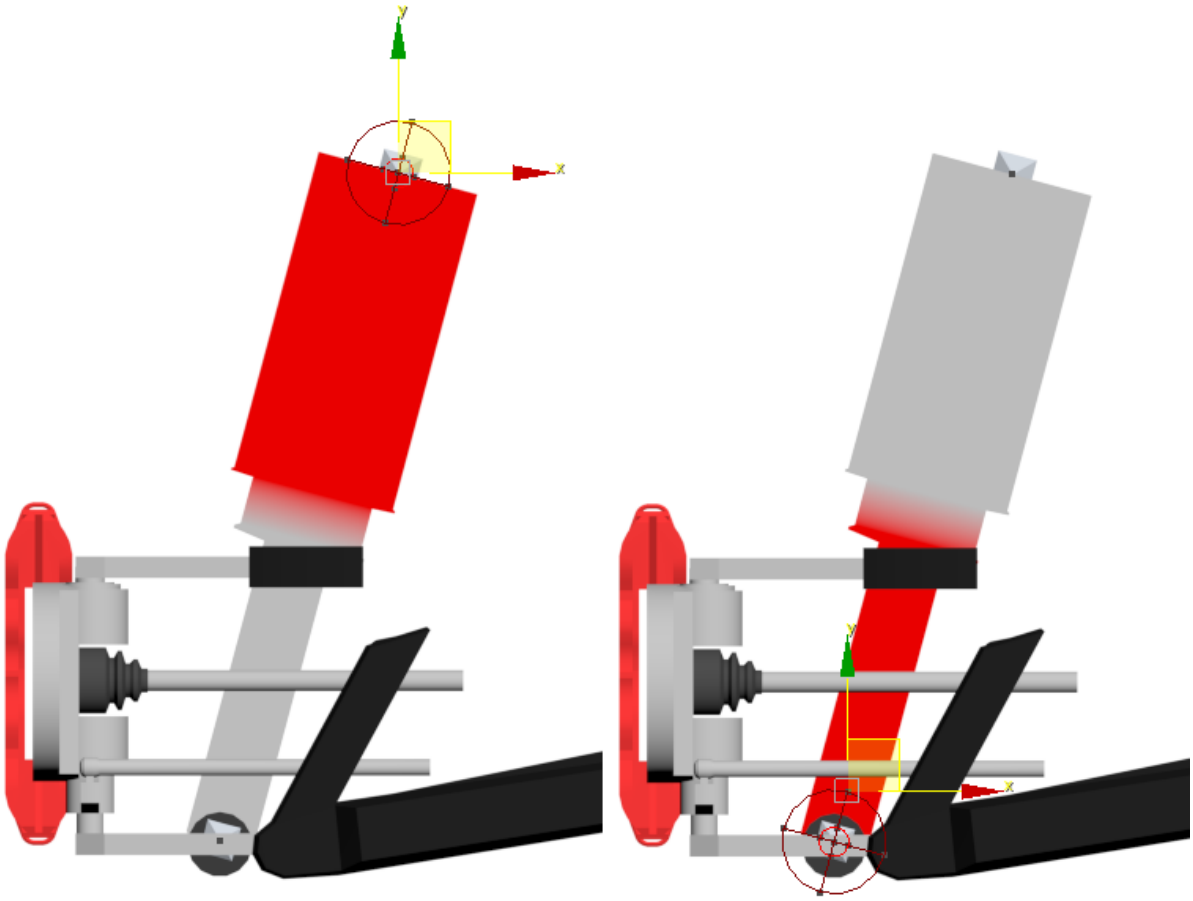
Když jsou kosti připraveny, je nutné je propojit s objekty. Tento proces je často označován jako rigování a v 3ds max k tomu slouží modifikátor `Skin`. Dále také platí, že pokud budeme v UE používat objekt, který má nastavené kosti, musí mít každý objekt, který má být zobrazen či jinak používán, přiřazenu alespoň jednu kost.

Jak už jsem zmínil při přípravě kostí, na autě je mnoho prvků, které zůstávají vždy na stejném místě, například karoserie, sedačky, skla. Všechny takové objektu jsou přiřazeny ke kosti `b_karoserie`. Tyto objekty půjdou následně v simulaci skrýt, aby bylo možné lépe zkoumat fungování nápravy. Na hlavní kosti `b_root` jsou přiřazeny pouze objekty pro uložení nápravy, jelikož ty by měly být zobrazené pořád. Spodní rameno je přichyceno ke kosti `b_pl_dolni_rameno_tehlice`, horní rameno podobně k `b_pl_horni_rameno_tehlice` a těhlice patří ke kosti `b_pl_tehlice_dolni_cep`. Kolo patří ke kosti `b_pl_kolo` a brzda podobně k `b_pl_brzda`. U přední nápravy jsem musel ještě nastavit tyč řízení, která patří ke kosti `b_pl_cep_rizeni`.

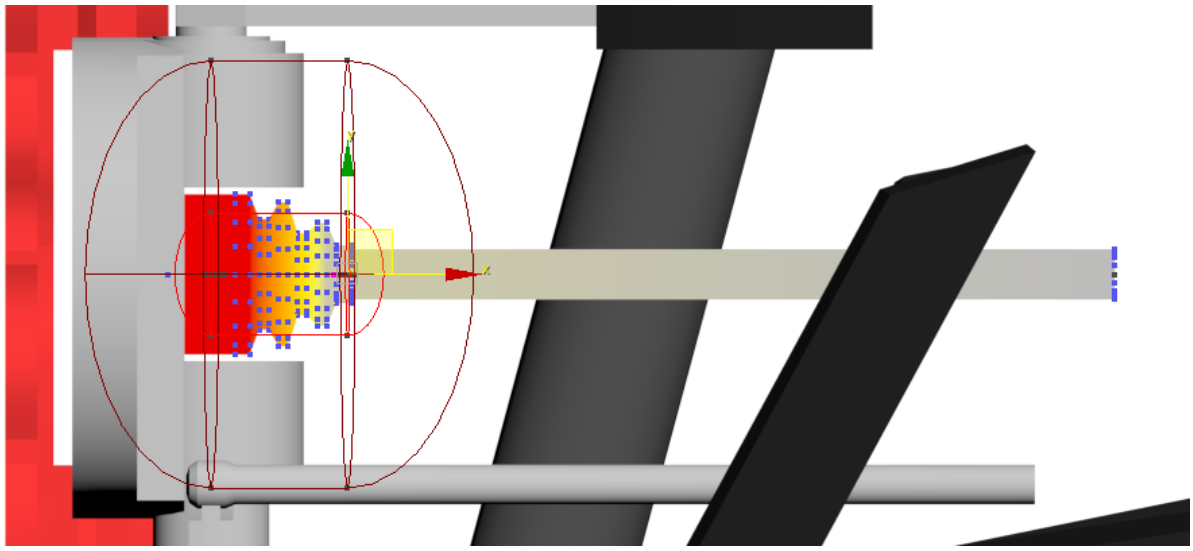
S poloosou a tlumičem je to trochu složitější. Tyto objekty musí mít přiřazeny 2 kosti, přičemž každá kost ovládá jinou část objektu. Tlumič bude mít přiřazeny kosti `b_pl_tlumic_dolni` a `b_pl_tlumic_horni`. Dále je nutné přesně specifikovat, které části modelu (tlumiče) jsou ovládány kterou kostí. To se dělá nastavením tzv. `Envelopes`, v češtině se také často používá označení váhy. Horní kost přímo ovládá horní část tlumiče s hliníkovou konstrukcí až po začátek manžety. Spodní část bude přímo ovládat zbytek bodů. Slůvko přímo říká, že daná váha bude mít maximální hodnotu, která činí 1.0 (což odpovídá 100% popřípadě červenou barvou při grafickém znázornění). Správné nastavení vah tlumiče si můžete prohlédnout na obrázku 11.

Poloosa je složitostí ještě trochu dále. Pokud bych udělal rozdělení vah podobně jako u tlumiče, docházelo by k nereálným deformacím. Na poloose je totiž manžeta, která sice mění svůj tvar v závislosti na natočení kol, pohyb na jejím začátku (ve směru od motoru) však bude jiný než na jejím konci. Proto jsem váhy rozdělil ve vhodném poměru každou pro každý jednotlivý ohyb manžety. To můžete vidět na obrázku 12 jako přechod z červené barvy (100%) přes žlutou (40%) až po šedou přímo u motoru (0%). Váhy pro opačnou kost `b_pl_poloosa_diferencial` jsou přesně opačné.

Také jsem přiřadil všechny části volantu kosti `b_volant` a ukazatele budíků odpovídajícím kostem.



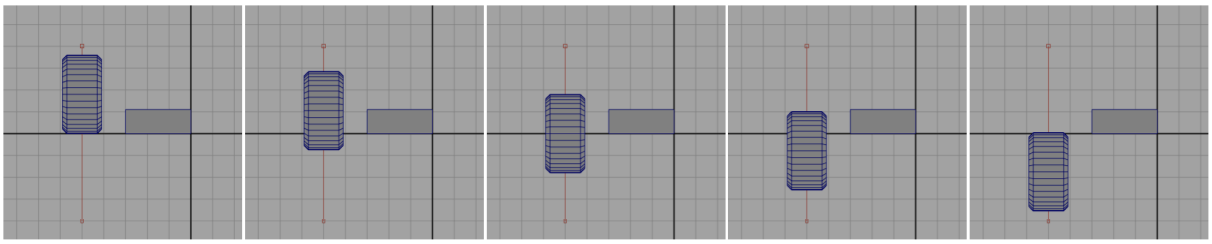
Obrázek 11: Váhy kostí pro tlumič, vlevo pro horní kost, vpravo pro dolní



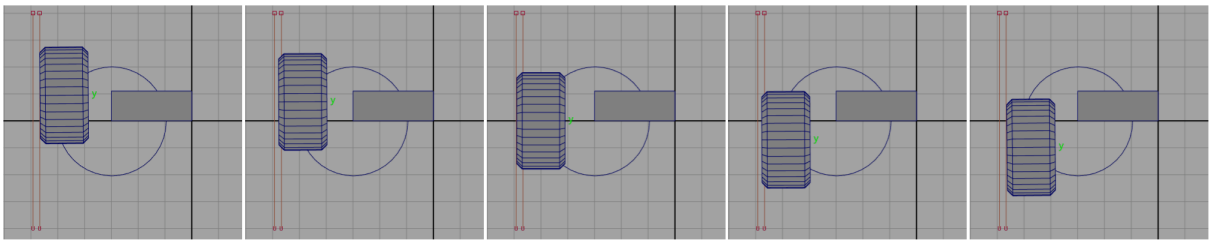
Obrázek 12: Váhy kosti b_pl_tehlice_stred pro poloosu, červené zvýraznění značí 100% ovládání danou kostí, žluté 40% a šedé 0% (čili kost nemá na toto místo žádný vliv)

5.2.2 Specifická příprava pro UE

Jelikož jsem se rozhodl využít již připravené funkcionality Unreal Enginu, bylo nutné zjistit, jakým způsobem připravit kosti, aby byl výsledek snadno použitelný právě v UE. Ve hrách je totiž běžně pohyb kola zjednodušen a probíhá pouze ve svislé ose, což znázorňuje obrázek 13. Ne jinak je tomu v UE. V reálném světě však kolo vykonává pohyb po kružnici, jak můžete vidět na obrázku 14. Já simulovaný pohyb z UE využiji, ne však k zobrazení kol, nýbrž pouze k naklápění spodního ramene a následně i transformaci dalších objektů. Proto je nutné mít ještě jednu kost pro každé kolo, kterou jsem pro přední levé kolo pojmenoval `b_pl_fyz` (jako fyzické kolo). Celou hierarchii kostí pro levé přední kolo můžete vidět na obr. 15.



Obrázek 13: Simulovaný pohyb kola, převzato a upraveno z [2]



Obrázek 14: Reálný pohyb kola, převzato a upraveno z [2]

5.2.3 Export modelu

Nyní je nutné model správně exportovat. Nejlépe podporovaný formát se z mé zkušenosti jeví FBX, proto jej používám a popíšu jeho nastavení. V defaultním nastavení (`Autodesk Media & Entertainment`) jsem v záložce `Geometry` vypnul možnosti:

- TurboSmooth, jelikož vyhlazování na patřičných místech mám již vyřešeno,
- Convert Deforming Dummies to Bones, aby nevznikaly nechtěné kosti, pokud nějaký objekt ovládá jiný.

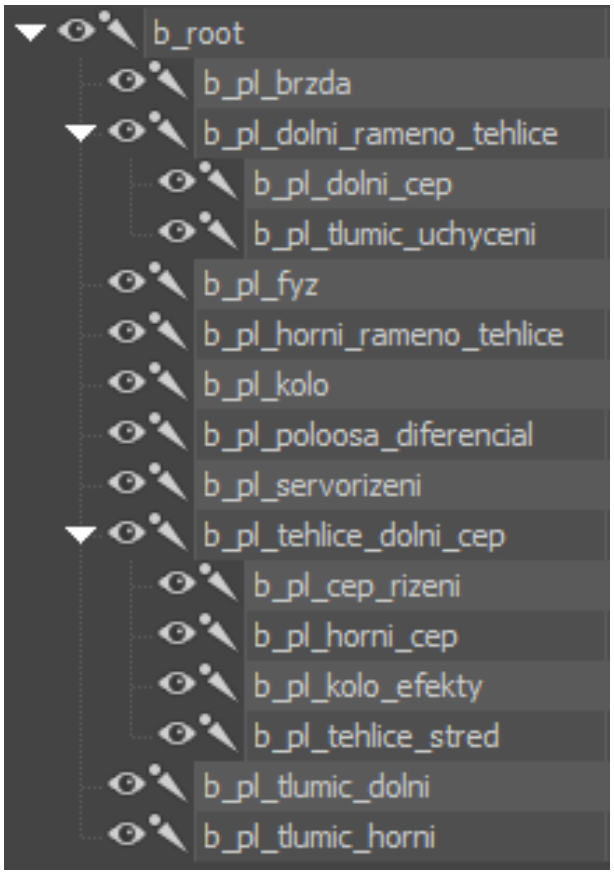
Naopak jsem ve stejné záložce zapnul:

- Smoothing Groups, protože využívám možnosti vyhlazení pouze části objektu,
- Tangents and Binormals, jelikož chci vyexportovat také UV souřadnice,

- Triangulate, aby byly všechny objekty převedeny na trojúhelníkovou síť.

Poslední změnou byla také změna verze FBX souboru z `FBX 2014/2015` na `FBX 2016/2017`, které se nachází v rámci `Advanced options`.

Kompletní model včetně domodelovaných součástí náprav se skládá ze 108 objektů, které jsou tvořeny celkem 87131 plochami, respektive 97529 vrcholy. Dalších 65 objektů tvoří kosti pro následnou animaci.



Obrázek 15: Kompletní hierarchie kostí pro levé přední kolo

6 Vytvoření projektu

Začal jsem vytvořením čistého C++ projektu s předvolbou Vehicle a zároveň bez startovacího balíčku. Projekt jsem pojmenoval HAJ0094, proto také tento název uvidíte i v dalších částech. C++ projekt proto, že část funkcionality bude provedena právě programováním ve zdrojovém kódu. Unreal Engine jsem měl v době psaní práce v poslední verzi, konkrétně 4.18.3. K vytvoření C++ projektu je nutné mít také nainstalováno buď Visual Studio nebo XCode, které se zároveň starají o kompilaci kódu. V mém případě se jednalo o Visual Studio 2017.

Po vytvoření projektu jsou vytvořeny 3 C++ třídy, týkající se samotného auta. Jedná se o `HAJ0094WheelFront`, `HAJ0094WheelRear` a `HAJ0094Pawn`. První dvě třídy dědí z třídy `Vehicle Wheel` a jsou, jak už název napovídá, určeny pro přední a zadní kola. Nastavují informace o rozměrech kol, zda-li je náprava řídící (a v jakém maximálním úhlu) a také informaci, zda-li ruční brzda ovládá daný typ kola. Poslední zmíněná třída `HAJ0094Pawn` dědí z třídy `WheeledVehicle`. V rámci konstruktoru této třídy je provedeno několik akcí, první z nich jsou nastavení `Skeletal Mesh` a animačního blueprintu. Poté jsou nastavena kola za použití odpovídající třídy pro přední nebo zadní kolo a nastavení pozice na předem vytvořenou kost s dodatečným posunutím. Pak jsou vytvořeny a nastaveny pozice kamer a nakonec i pozice textů pro zobrazení rychlosti a zařazeného převodového stupně. Další metody pak řeší zejména ovládání vozidla včetně používání ruční brzdy či přepínání vytvořených kamer. Plně připravenou C++ třídu je možno rovnou umístit do mapy a použít, není nutné z ní vytvářet blueprint.

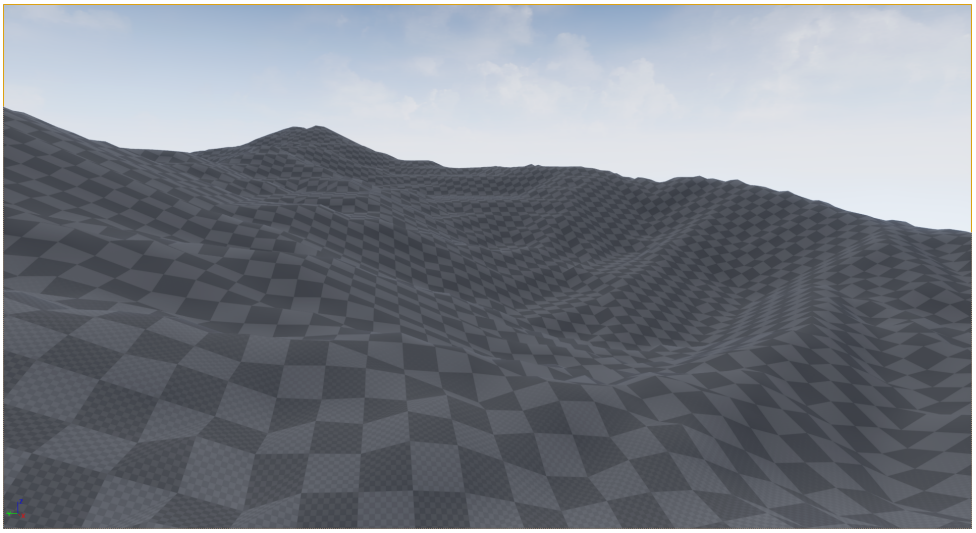
Z C++ třídy je možno vyrobit blueprint a rozšířit tak funkcionalitu dále. Vhodným nastavením proměnných (pomocí `UPROPERTY`) je možno například zabránit nechtěným změnám určitých proměnných, takže v blueprintu budou pouze ke čtení apod. Takové nastavení je využito například pro standardní kamery, které z blueprintu (bez změny C++ třídy) nejdou odstranit. Naopak je ale možné upravit jejich umístění, rotaci a jiné.

Rozšíření existujícího blueprintu C++ kódem však možné není, proto je dobré brát tuto informaci v potaz již při návrhu.

7 Tvorba terénu

Aby bylo možné simulaci někde provést, je potřebný nějaký terén či povrch, po kterém bude moci auto jezdit. Ten by se měl, jako každá reálná testovací trať, skládat z několika typů povrchu, které budou ovlivňovat chování auta.

Na tvorbu povrchu je vhodné použít předdefinované funkce `Landscape`, což jsem také udělal. Vytvořil jsem si nový povrch a zejména použitím modifikátorů `Sculp`, `Erosion` a `Smooth` jsem vyrobil na povrchu nerovnosti. `Scuplt` slouží zjednodušeně ke zvednutí či zmenšení části plochy, ať už s plynulým přechodem či téměř ostrou hranou. `Erosion` provede nad zvolenou částí terénu simulaci eroze půdy. Aby byl výsledek o něco reálnější a změny povrchu po erozi byly uhlazenější, použil jsem ještě modifikátor `Smooth`. Využití těchto modifikátorů ukazuje obrázek 16.



Obrázek 16: Zvrásnění terénu

Dále jsem vytvořil materiál, který se skládá ze 4 `Layer Blend` vrstev:

1. Asfalt
2. Šotolina
3. Tráva
4. Skála.

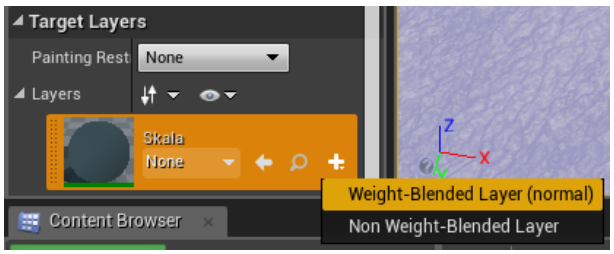
Textury povrchů jsem získal z webu textures.com [17]. Normálové mapy k získaným texturám se na serveru nenacházejí, musel jsem je vytvořit sám, k čemuž jsem použil online nástroj [18]. Nastavení tvorby normálových map jsem nechal v základním nastavení, což později znamenalo nutnost snížení intenzity normálové mapy v UE, jelikož povrch již vypadal nepřirozeně. Zároveň jsem pro terén asfaltu zkombinoval dvě textury dohromady pomocí lineární interpolace, přičemž

obě textury mají UV souřadnice škálovány jinou hodnotou, aby nedocházelo k příliš znatelnému opakování textur.

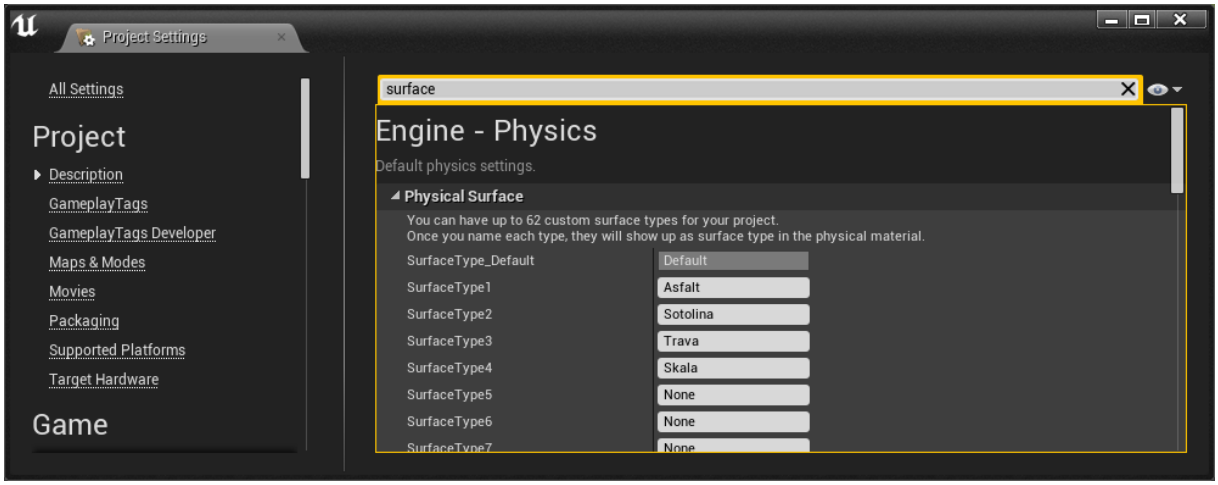
Po přiřazení tohoto materiálu k vytvořenému terénu se v záložce `Paint` objeví volby `Layers`. Pro každou `Layer Blend` vrstvu z předchozího kroku jsem vytvořil `Layer Info` objekt typu `Weight-Blended Layer (normal)`. Postup znázorňuje obrázek 17. To následně umožní kreslením nastavit na určité místo povrchu požadovanou texturu.

Abych však mohl univerzálně rozeznat, na jakém typu povrchu se auto, respektive kolo nachází, bylo nutné nastavit typy povrchu tzv. `Surface Type`. V `Project Settings` v záložce `Engine` a `Physics` jsou definovány typy povrchu, standardně jsou `None` kromě implicitního. Upravil jsem tedy typy povrchu dle mých požadavků, což znamenalo vyplnění položek `Surface Type 1-4` dle mých materiálů, to znázorňuje obrázek 18. Poté jsem pro každý typ materiálu vytvořil blueprint typu `Physical Material`. V nastavení každého takového materiálu jsem pak nastavil odpovídající `Surface Type` a tento materiál jsem přiřadil také odpovídajícím `Layer Info` objektu, který má v nastavení parametr `Phys Material`.

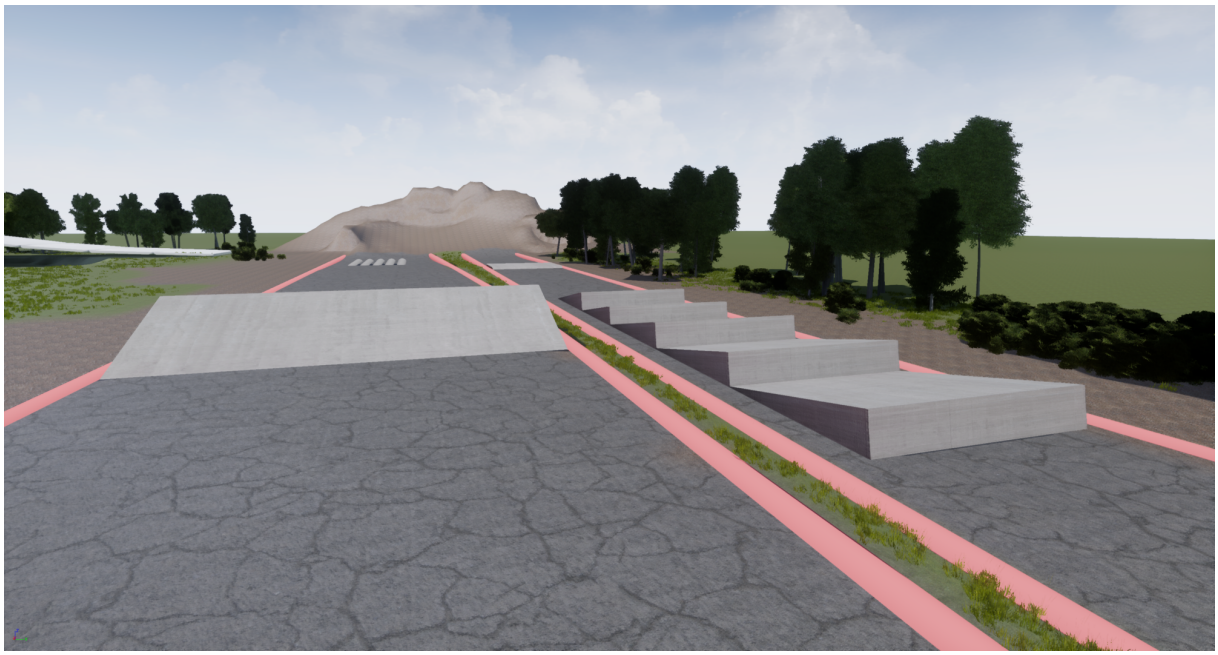
Použití fyzikálních materiálů mimo jiné umožňuje také nastavení jiného koeficientu tření, čehož jsem i využil. Na závěr jsem do terénu ještě přidal betonové překážky a mantinely. Aby terén nepůsobil příliš prázdným dojmem, přidal jsem také efekty trávy, keřů a stromů pomocí `Foliage`. To je systém v UE, který umožňuje přidání právě takových doplňkových objektů, které by však měly přidávat pouze minimální náročnost na systémové prostředky. Objekty jsem získal z volně dostupného balíčku od vývojáře UE [19]. Na obrázku 19 je možno vidět část trasy s překážkami a doplňkovými objekty.



Obrázek 17: Vytvoření záznamové vrstvy pro typ povrchu



Obrázek 18: Nastavení vlastních jmen pro typy povrchů



Obrázek 19: Terén s překážkami a doplňkovými objekty

8 Vlastní C++ třídy

Některé části se mi jevily přehlednější a vhodnější řešit doprogramováním vlastní funkcionality. Unreal Engine umožňuje rozšiřování díky C++, proto jsem také v kapitole 6 vytvořil projekt typu C++ a ne blueprint. Výhodou je, že Unreal Engine má, dle mého názoru, skvělou integraci s Visual Studiem. Jelikož však C++ třída může rozšiřovat pouze jinou C++ třídu a ne blueprint, je nutné začít potřebnými změnami právě v C++.

Protože má testovací trať zahrnuje více typů povrchu a například efekt na šotolině není stejný jako na asfaltu, vytvořil jsem pomocnou třídu `NastaveniEfektu`, založenou na typu `DataAsset`. Hlavičkový soubor je přiložen v příloze C, zdrojový v příloze D. Tato třída v sobě uchovává požadovaný efekt pro jednotlivé typy povrchu a informaci, zda se má efekt zobrazovat pouze při smyku nebo vždy. Například u asfaltu se objeví kouř od pneumatik pouze při smyku, kdežto při projíždění šotolinou se bude vznášet oblak prachu za autem téměř vždy. Pro zlepšení čitelnosti kódu jsem také zavedl pro typy povrchu konstanty, které nahrazují používání předdefinovaného, nic neříkajícího, názvu. `NastaveniEfektu` jsem také rozšířil o nastavení stop pneumatik, které je použito v dalších částech této práce. Konkrétně se jedná, podobně jako u efektů, požadovaný materiál stopy a informaci, zda se má efekt zobrazovat pouze při smyku, samozřejmě pro každý typ povrchu.

Po vytvoření projektu je k dispozici mimo jiné vytvořena třída `HAJ0094Pawn`, která obsahuje funkce týkající se auta a jeho ovládání. Jelikož mám pomocí C++ implementovány efekty, vytvořil jsem novou třídu `HAJ0094_efekty`, která dědí právě z `HAJ0094Pawn`. To umožňuje provést pouze mnou požadované změny či rozšíření, aniž bych se musel složitě orientovat v ne zrovna malé původní třídě. Zároveň je také snadno viditelná moje práce. Hlavičkový soubor je přiložen v příloze E, zdrojový v příloze F.

Třída `HAJ0094_efekty` přidává 3 metody a rozšiřuje jednu jedinou, `Tick`. První nová metoda se jmenuje `float VratRychlostAuta() const` a není třeba ji dále popisovat. Další nová metoda je `void InicializaceEfektu(int kolo)`, která vytvoří instanci správného efektu a umístí zdroj efektu na pozici kola. Celočíselný parametr `kolo` určuje, o které kolo se jedná. Poslední nová metoda je `void AktualizaceEfektu()`. Ta je oproti předchozím metodám podstatně delší a řeší správné nastavení požadovaných efektů a jejich zapínání či vypínání. Pokud je rychlost auta nízká, efekty se vypnou, jelikož při příliš pomalé jízdě nevzniká prach ani na šotolině. Pokud je rychlost větší než zvolený práh, metoda dále zjistí, po jakém materiálu se kolo pohybuje a podle něj si zjistí požadovaný efekt. Je-li aktuální efekt kola jiný než požadovaný, je aktuální efekt deaktivován, odstraněn a přepsán nově vytvořenou instancí požadovaného typu efektu. Následně už pouze zjistím, zda-li má být efekt zobrazen pouze při smyku a podle toho jej zapnu, popřípadě vypnu.

8.1 Vytvoření efektů

V tomto kroku jsem také vytvořil odpovídající efekty pro jednotlivé typy povrchů. Pro asfalt se jedná o efekt kouře pouze při smyku vozidla. Pro trávu jsem vytvořil efekt odlétávajících kousků trávy a jízda po šotolině způsobuje nejen prach za vozidlem ale také odlétávající kamínky.

Pro efekt prachu jsem vytvořil texturu kruhového gradientu z bílé barvy uprostřed po černou v okrajích, která slouží jako základ nově vytvořené materiálu, kterému jsem nastavil `Blend Mode` na `Translucent` a `Shading Model` na `Unlit`. Aby nebyl vidět pouze efekt jednolitého kruhu, přidal jsem k mapě průhlednosti také vhodně nastavený uzel `Noise`, který mi do mapy průhlednosti přidal nepravidelnou strukturu. Abych mohl materiál využít i pro efekt prachu, který má pouze jinou barvu a hustotu (což je opět průhlednost), přidal jsem také parametr `Particle Color`, který mi právě toto umožňuje. Pro efekt odlétávajícího kamení jsem vytvořil náhodný mnohoúhelník, v mém případě šestiúhelník, a i jej jsem vybavil stejnou parametrizací `Particle Color`.

Nyní jsem z připravených materiálů vytvořil požadované efekty pomocí `Particle System`. Pro asfalt jsem vytvořil částicový systém pouze s jedním emiterem pro kouř. Tomu jsem nastavil, ať se efekt objevuje v různých počátečních velikostech v rámci určitého rozmezí, po dobu zobrazení efektu se zvětšuje jeho velikost a zároveň také mění složka průhlednosti. Vcelku jednoduchým nastavením jsem dosáhl přirozeného chování. Efekt šotoliny má stejný emiter pro kouř jako efekt asfaltu, má však mírně pozměněné parametry (hustota, barva, aj). Šotolina má však ještě jeden emiter pro odlétávající kamínky. Ten má navíc nastavení počáteční rychlosti `Initial Velocity` (aby kamínky od kol opravdu odlétávaly), ale zase je každý kamínek gravitací tažen dolů, což zajišťuje vhodné nastavení modulu `Acceleration`. Tráva využívá opět stejné moduly emiteru jako kameny v případě šotoliny, je však zásadně upraveno škálování emitovaných objektů a jejich barva, čímž jsem dosáhl efektu odseknutých stonků. Moduly všech 3 efektů jsou vidět na obrázku 20.

Obrázek 20: Moduly částicových systémů pro asfalt, šotolinu a trávu

9 Sdílený blueprint pro auta

Z každé C++ třídy může být vyroben blueprint, který může dále rozšiřovat funkcionalitu. Já jsem se rozhodl vytvořit blueprint, ve kterém bude implementována obecná funkcionalita auta. Tím mám na mysli například pohyb či vytváření stop za vozidlem, jelikož není důvod toto řešit pro každé auto zvlášť. V mé práci je ve výsledku sice auto pouze jedno, tímto návrhem jsem však umožnil snadné a rychlé přidání dalších aut, stačí pro každý další model dodržet stejné pojmenování kostí jako v kapitole 5.2. Vytvořil jsem tedy z mé C++ třídy `HAJ0094_efekty` blueprint `AutoSEfekty`, ve kterém bude implementována obecná funkcionalita auta.

V tomto blueprintu jsem přednastavit jména kostí všech kol. To se nachází v komponentě `VehicleMovement`, přesně pak v `Vehicle Setup` a následně `Wheel Setups`. Jména kostí (`Bone Name`) jsme volil v pořadí:

1. `b_pp_fyz`
2. `b_pl_fyz`
3. `b_zp_fyz`
4. `b_zl_fyz`

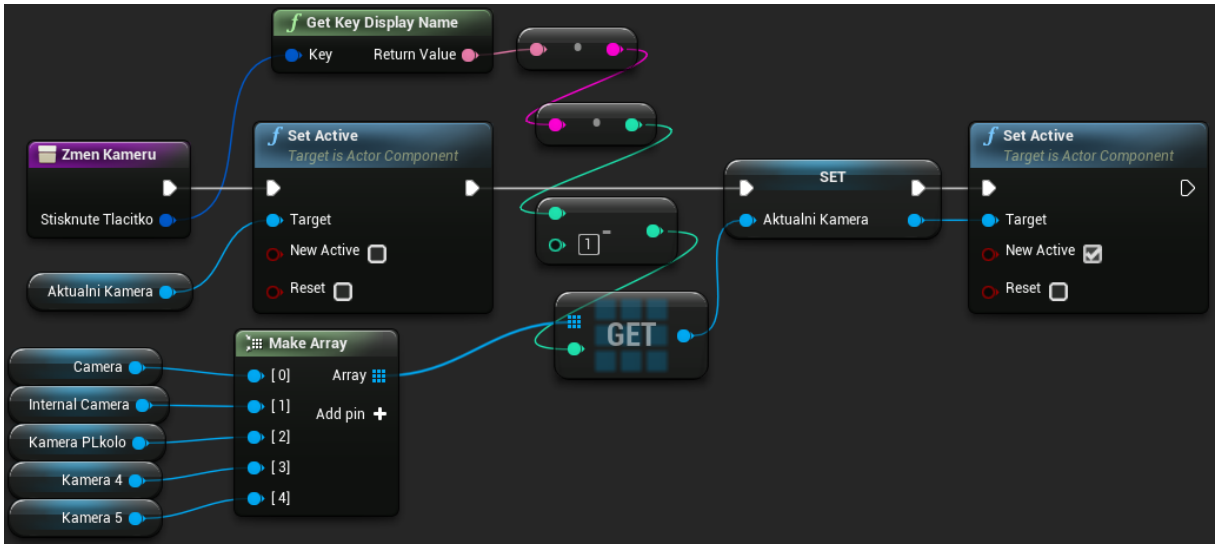
9.1 Změna kamery

Změna pohledů je jedna ze základních možností her či podobných simulací. Implicitně jsou v blueprintu auta 2 kamery, venkovní a z interiéru. Já jsem například přidal další k detailnímu sledování pohybu náprav. Vybral jsem si implementaci přepínání pohledů podle čísla kamery. Nejprve je však nutné si zvolit jednu kameru, která bude použita po spuštění. Tato kamera bude mít zapnutou funkci `Auto Activate`, u ostatních kamer jsem tuto možnost vypl. Dále jsem si přidal jednu proměnnou `AktualniKamera` typu `Camera Component`, ve které budu udržovat informaci o aktuálně nastavené kameře. Tu je zároveň také nutné na začátku inicalizovat, což jsem provedl v rámci vykonávání události `BeginPlay`.

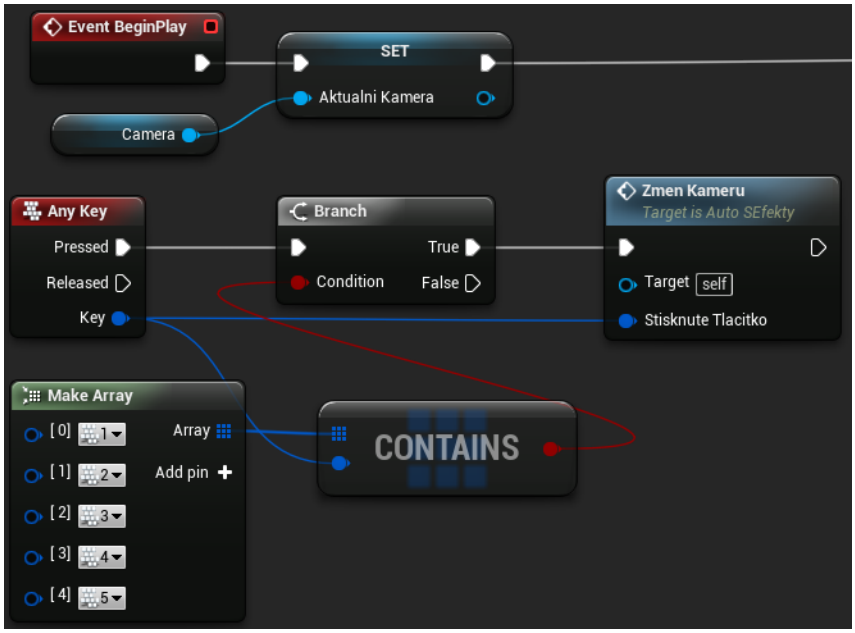
Dále jsem vytvořil metodu `ZmenKameru` (viz. 21), která má jeden vstupní parametr, stisknutou klávesu. Po zavolání metody se nejprve provede deaktivace aktuální kamery. Hned poté se interně vytvoří pole všech kamer a podle stisknutého tlačítka se vybere odpovídající kamera z onoho pole. Je-li stisknuta například klávesa 2, z pole se vybere druhá položka (tedy index 1). Vybraná kamera je uložena do proměnné `AktualniKamera`, vzápětí je i aktivována a metoda končí.

Ještě je však nutné metodu zavolat, avšak pouze v relevantních případech. Proto jsem přidal do `Event Graph`u tohoto blueprintu také kontrolu na stisk jakékoliv klávesy. Po stisku je ověřeno, zda došlo ke stisku očekávané klávesy (v mém případě jsou to klávesy 1-4, respektive tlačítka s diakritikou na české klávesnici). Pokud je podmínka splněna, je metoda zavolána a předáno

stisknuté tlačítko. Toto rozhodnutí i inicializace při události `BeginPlay` můžete vidět na obrázku 22.



Obrázek 21: Metoda ZmenKameru



Obrázek 22: Inicializace standardní kamery a kontrola stisknutých tlačítek

9.2 Skrytí částí modelu vozidla

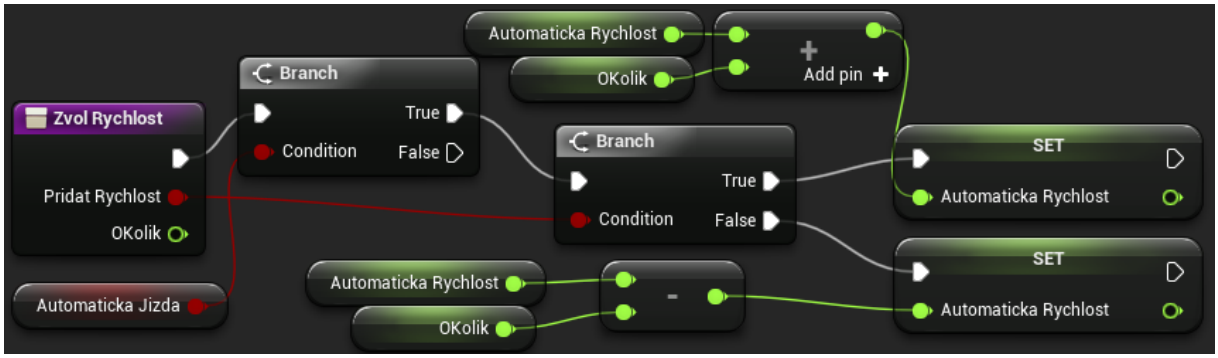
Pro lepší názornost je také možnost vypnout zobrazení pevných částí karoserie, jako jsou plechy, interiér, skla apod. V takové chvíli zůstanou viditelné pouze nápravy, volant a budíky. Přepínání se provádí stiskem klávesy `O`. Po stisku je zavolána akce `Hide Bone by Name`, která má jako

parametr zvolenu kost `b_karoserie`, kterou jsem přesně za tímto účelem vytvořil dříve. Ukázka takového režimu je vidět v příloze A.

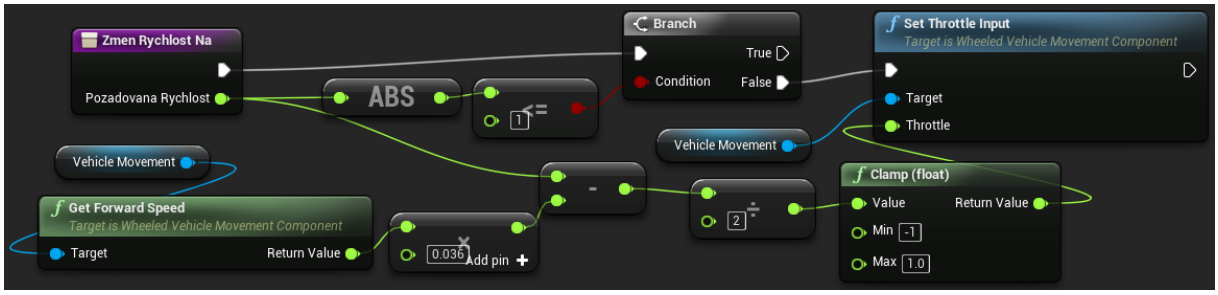
9.3 Nastavení tempomatu

Jelikož se pouze pomocí klávesnice špatně udržuje požadovaná rychlost, přidal jsem do blueprintu tempomat. Jelikož vzájemné použití tempomatu a ovládání pomocí šipek se vylučuje, vytvořil jsem si bitovou proměnnou `AutomatickaJizda`, která udržuje informaci o aktuálním režim ovládání. Dále jsem vytvořil proměnnou typu `float` se jménem `Automati ckaRychlost`, pro uchovávání požadované rychlosti.

K nastavení požadované rychlosti jsem vytvořil metodu, která změní nastavenou rychlost o předdefinovaný počet km/h (standardně 5 km/h), viz obrázek 23. K rozhodnutí, zda-li se má rychlost zvětšit nebo zmenšit, slouží bitový parametr `PridatRychlost`. Zároveň jsem zařídil, aby byla tato funkce s odpovídajícím parametrem zavolána na stisk klávesy `8` a `5`, tentokrát však nutno na numerické klávesnici. Nastavení požadované rychlosti je provedeno, ještě je ale potřeba zajistit dosáhnutí této rychlosti, potažmo její udržování. K tomu slouží metoda `ZmenRychlostNa`, kterou je možno vidět na obrázku 24. Pokud je absolutní hodnota požadované rychlosti menší nebo rovna 1 km/h, nedělá metoda nic. Pokud je však větší, metoda automaticky nastaví odpovídající množství plynu s tím, že požadované množství plynu je pro zmírnění prudkosti změn mírně upraveno.



Obrázek 23: Metoda ZvolRychlost

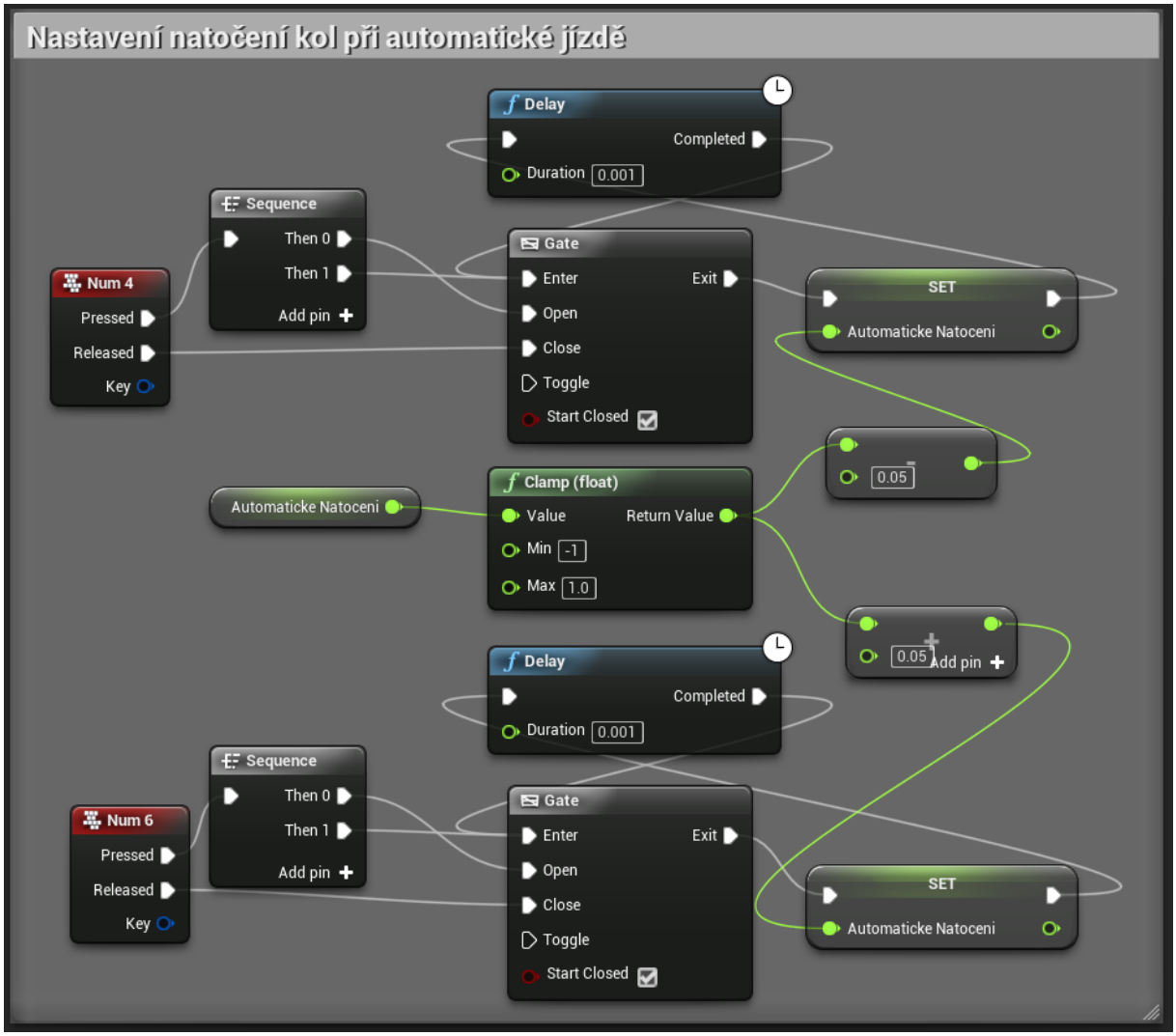


Obrázek 24: Metoda ZmenRychlostNa

9.4 Udržování natočení kol

Podobně jako je možno nastavit požadovanou rychlost, přidal jsem i možnost nastavit natočení kol, které bude udržováno. K udržování hodnoty jsem opět vytvořil novou proměnnou, `AutomatickeNatoceni`. Jelikož je dobré mít nastavení natočení kol plynulé, ovšem bez nutnosti opakování stisku tlačítek, je konstrukce o něco složitější, viz obrázek 25. Použil jsem uzel `Gate`, který slouží k provádění kódu ve větvi `Exit`, je-li brána ve stavu `Open`. Po stisku tlačítka je brána otevřena, při uvolnění tlačítka zase zavřena. Pro rozumnou citlivost se mi osvědčilo zpoždění mezi dalším krokem natočení 0.001 vteřiny. To se sice může zdát jako malá časová prodleva, natočení kol však probíhá po 5% z maximálního povoleného natočení kol (definováno v odpovídající třídě typu `VehicleWheel`).

Oproti tomu funkce `ZmenNatoceniKolNa` je pro změnu velice krátká, ta pouze nastaví natočení kol na hodnotu nastavené proměnné.



Obrázek 25: Plynulé natáčení kol při stisku tlačítka

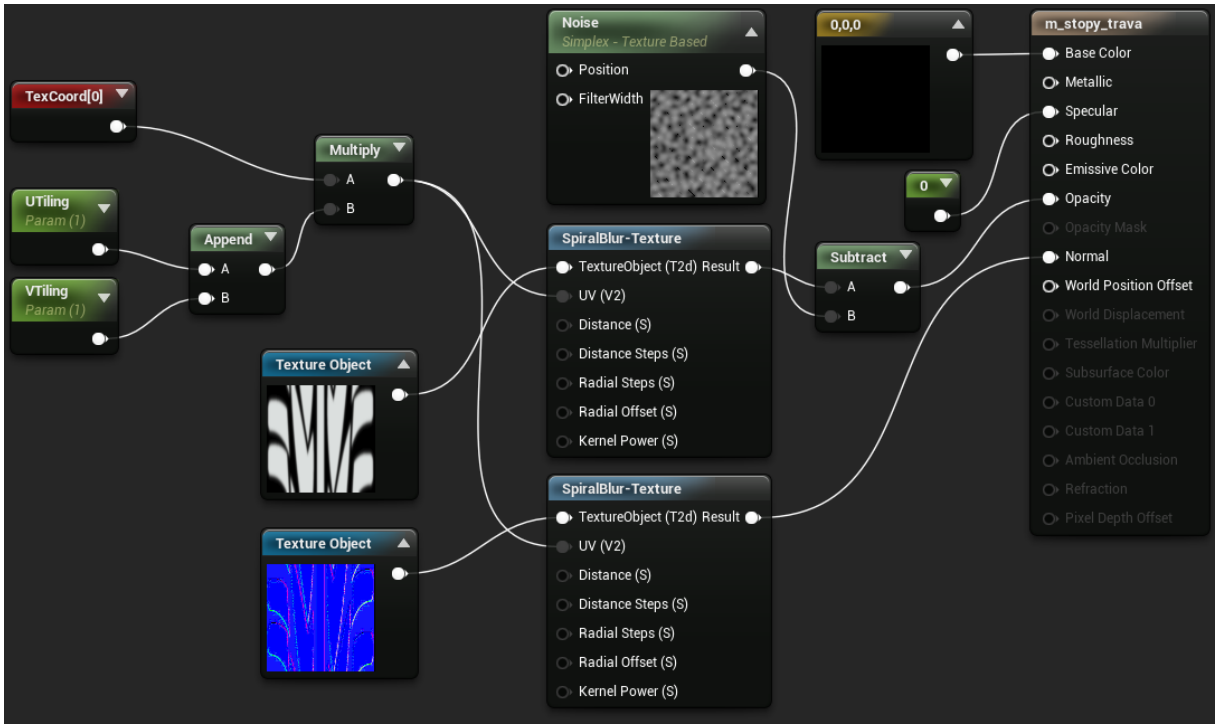
9.5 Zanechávání stop pneumatik

Jedním z cílů této práce bylo také zanechávání stop za jedoucím vozidlem. Nejprve jsem vytvořil texturu zanechávaných stop, viz obrázek 26. Při jejím vytváření je nutné dbát na to, aby byla textura opakovatelná bez viditelných přechodů. Stejnou strukturu vzorku jsem poté i vymodeloval v 3ds max, aby zanechávané stopy korespondovaly s modelem.

Obrázek 26: Textura zanechávaných stop

9.5.1 Vytvoření materiálu

Po vytvoření nového materiálu je nutné změnit `Material Domain` ze `Surface` na `Deferred Decal` a `Blend Mode` z `Opaque` na `Translucent`. Nastavení uzlů materiálu je možno vidět na obrázku 27. Je vhodné zmínit dva parametry, `UTiling` a `VTiling`, vysvětlení jejich použití bude následovat později. Materiál z obrázku využívá i uzel `Noise`, díky kterému bude mít zanechávaná stopa méně či více výraznější místa. Ostatně v reálném světě také nemá stopa kol všude stejnou intenzitu. Tyto stopy jsem nastavil pro povrch trávy a šotoliny. Pro asfalt a skálu jsem vyrobil kopii tohoto materiálu avšak bez uzlu `Noise`.



Obrázek 27: Materiál pro zanechávané stopy

9.5.2 Generování stop

Na trávě by měly být stopy zobrazeny pořád, čili musí být zajištěno přidávání stop pro každý kousek pohybu. K tomu se hodí událost `Tick`. Je však nutné si uvědomit, že tato událost je spouštěna v určitých intervalech, které se můžou lišit v závislosti na použitém hardwaru, nastavení detailů apod. Zároveň se také mění rychlost auta, pročež vyšší rychlost vozidla znamená větší uraženou vzdálenost, takže by tak mezi jednotlivými stopami vznikala prázdná místa. Proto jsem vymyslel způsob, který využívá události `Tick` ale zároveň je nezávislý na výkonu počítače či rychlosti auta. Zjednodušeně spočívá v tom, že porovnám aktuální pozici kola s pozicí kola v minulé události a podle uražené vzdálenosti umístím správně zvětšenou stopu s vhodně škálovanou texturou. Pro vytváření stop bylo potřeba připravit několik proměnných k ukládání nutných informací. Pro větší přehlednost jsem je vypsal do tabulky 1.

Jméno	**Typ**	**Defaultní hodnota**	**Popis**
KostiEfektu	pole `Name`	`["b_pp_kolo_efekty", "b_pl_kolo_efekty", "b_zp_kolo_efekty", "b_zl_kolo_efekty"]`	Jména kostí efektů, které využiji ke zjištění průsečíku kola s terénem.
PredchoziLokaceVK	pole `Vector`	`[(0,0,0), (0,0,0), (0,0,0), (0,0,0)]`	Čtyřprvkové pole souřadnic poloh průsečíků kola a terénu všech kol při posledním generování stop.
PredchoziRotaceVK	pole `Float`	`[0.0, 0.0, 0.0, 0.0]`	Čtyřprvkové pole rotací všech kol při posledním generování stop.
PredchoziNatoceniVK	pole `Float`	`[0.0, 0.0, 0.0, 0.0]`	Čtyřprvkové pole natočení všech kol při posledním generování stop.
ZpracovavaneKolo	`Integer`	`0`	Index zpracovávaného kola, používaný v cyklu. Bez této proměnné se dá obejít ale značně zlepšuje čitelnost kódu, respektive blueprintu.
Decal	`Material Instance (Object reference)`	`None`	Instance materiálu stopy.
DecalPouzePriSmyku	`Boolean`	`False`	Nese informaci, zda-li se mají stopy zobrazovat pouze při smyku.
Smyk	`Boolean`	`False`	Nese informaci, zda-li je auto ve smyku, což značí použití jiného nastavení Decalu.
AktualniLokace	`Vector`	`(0,0,0)`	Aktuální poloha průsečíku zpracovávaného kola a terénu.
AktualniRotace	`Float`	`0.0`	Aktuální rotace zpracovávaného kola.
AktualniNatoceni	`Float`	`0.0`	Aktuální natočení zpracovávaného kola.
UrazenaVzdalenost	`Float`	`0.0`	Uražená vzdálenost kola od posledního generování stopy.
ZivotnostDecalu	`Float`	`20.0`	Interval ve vteřinách, za jak dlouho stopa zmizí.

Tabulka 1: Potřebné proměnné pro správné generování stop

Jako první si zjistím, zda-li se auto nachází ve smyku či nikoli. To provádím porovnáním úhlu mezi aktuálním směrem pohybu vozidla s vektorem směřujícím vpravo od vozidla.s Dalším krokem je cyklus `ForEachLoop` přes všechna kola. První krok v rámci cyklu je nastavení proměnných `ZpracovavaneKolo`, `AktualniRotace` a `AktualniNatoceni`. Po nastavení proměnných zjišťuji, zda-li kola leží na povrchu. K tomu využívám funkci `LineTraceByChannel`, která potřebuje znát souřadnice 2 bodů, mezi kterými hledá onen průsečík. K tomu využívám pozici kosti primárně určené pro efekty, přičemž jednu souřadnice zvýším o 5cm v ose Z, druhou pro změnu snížím.

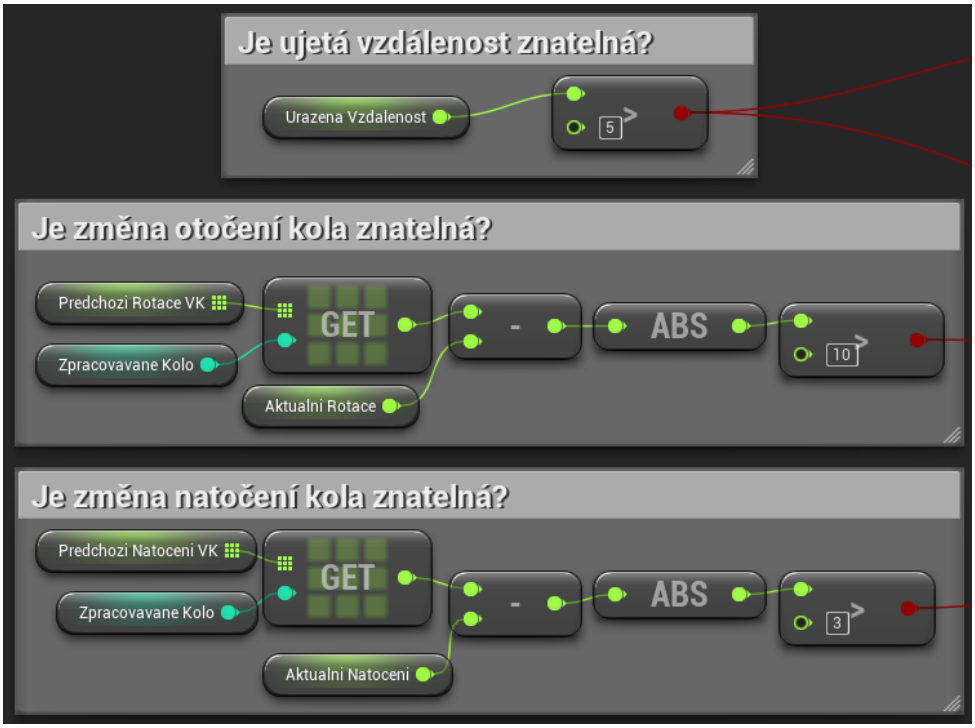
Pokud průsečík existuje, zjistím si typ terénu v bodě průsečíku a podle něj vytvořím odpovídající dynamickou materiálovou instanci, kterou uložím do proměnné `Decal`. Požadovaný typ stop beru z třídy `NastaveniEfektu`, kterou jsem popisoval v kapitole 8. Zároveň si také z této třídy beru informaci, zda se mají stopy zobrazovat pouze při smyku. Rovněž i tuto informaci si uložím, konkrétně do `DecalPouzePriSmyku`. Pozici průsečíku si uložím do proměnné `AktualniLokace` a vypočítám uraženou vzdálenost, která není nic jiného než délka vektoru rozdílu aktuální lokace průsečíku a průsečíku z minulé události `Tick`. A právě pozici minulých průsečíků mám v poli `PredchoziLokaceVK`.

V dalším kroku je nutné rozhodnout, zda-li se stopy mají kreslit a pokud ano, tak jestli stopy smyku či standardních stop. Stopy smyku se kreslí vždy při smyku. Normální stopy se kreslí, pokud:

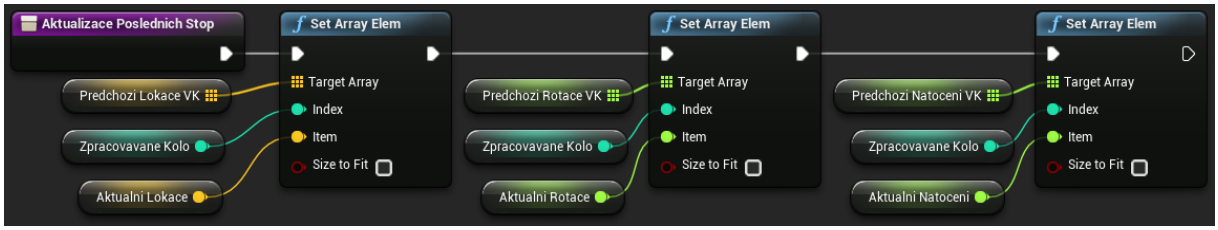
- `DecalPouzePriSmyku` proměnná není ve stavu `True` a uražená vzdálenost je znatelná (podle zvoleného prahu),
- při prokluzu kola při stání či prudké akceleraci,
- při natáčení kol přední nápravy při stání.

Porovnání uražené vzdálenosti, detekci prokluzu a natáčení kol můžete vidět na obrázku 28. Pokud se stopy kreslit mají, ale uražená vzdálenost je nižší než zvolený práh a nedošlo ani ke změně rotace či natočení kola, nedělá se nic a v další iteraci `Tick` se opět vzdálenost kontroluje, není-li již větší. V ostatních případech jsou vždy aktualizována pole `PredchoziLokaceVK`, `PredchoziRotaceVK` a `PredchoziNatoceniVK`. Jelikož se jedná o opakující sekvenci, vytvořil jsem funkci `AktualizacePoslednichStop`, kterou můžete vidět na obrázku 29.

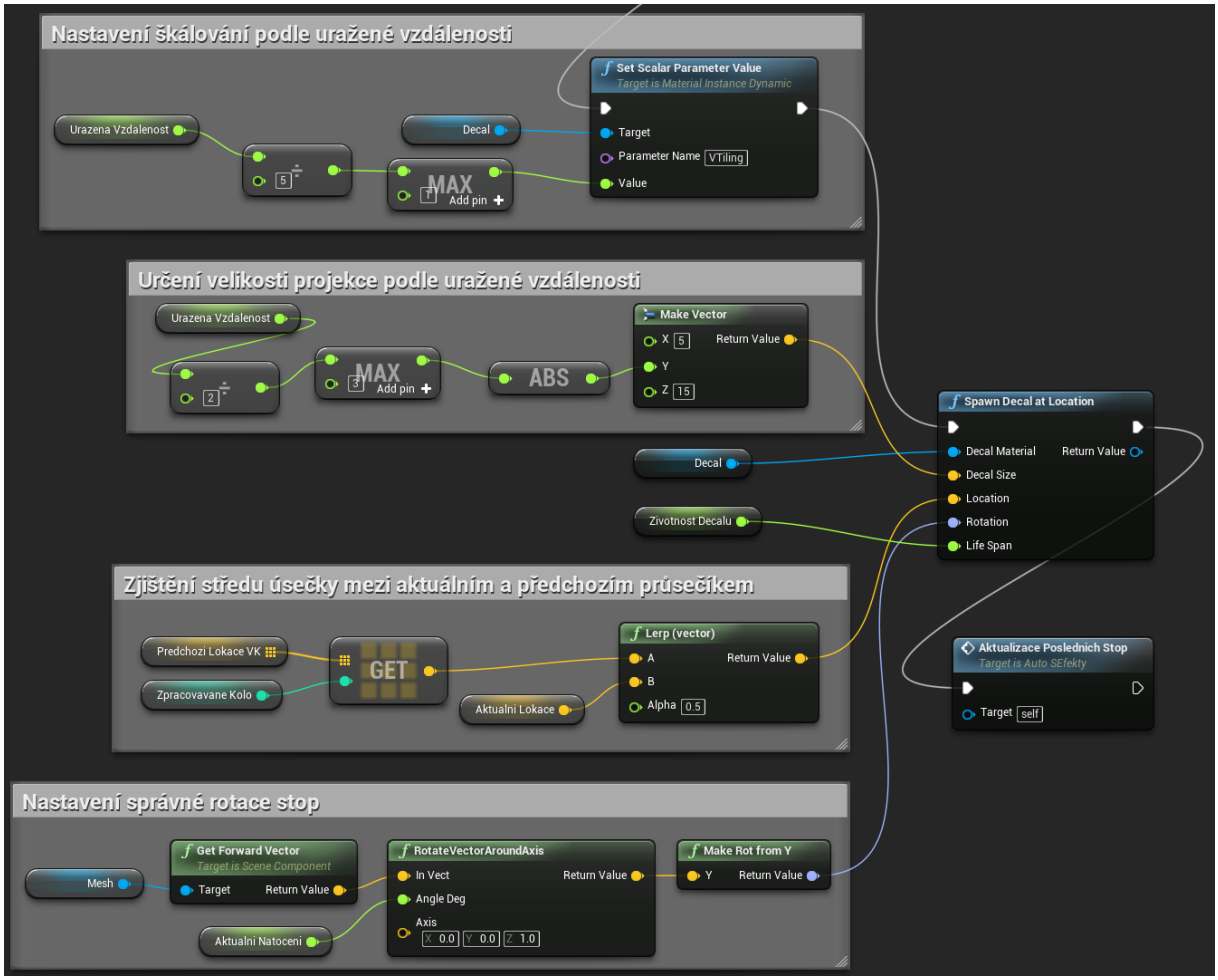
Generování stop probíhá vždy v polovině úsečky mezi aktuálním průsečíkem povrchu a předchozím průsečíkem. Zároveň se také velikost stop nastaví podle délky této úsečky. Jelikož se škáluje velikost stopy, je nutné také správně škálovat texturu, k čemuž slouží právě proměnné `UTiling` a `VTiling`. Jelikož však šířka stop zůstává vždy stejná, nastavuji pouze `VTiling`. Dále je nutné nastavit rotaci stop, kterou nastavuji podle aktuálního natočení kola. A poslední parametr je doba, po jaké stopy mizí. Pro sjednocené mizení stop jsem použil právě proměnnou `ZivotnostDecalu`. Tuto část generování stop vozidla znázorňuje obrázek 30. Odlišnost pro stopy smyku jsou dvě, první z nich je, že natočení stop není podle natočení kola, nýbrž podle směru



Obrázek 28: Podmínky kreslení stop



Obrázek 29: Aktualizace informací o posledních generovaných stopách



Obrázek 30: Generování běžných stop vozidla

pohybu vozidla. Druhá odlišnost je pak použití velice malého `VTiling` pro materiál, což místo standardní textury vykreslí pouze její roztažený zlomek (což vypadá jako efekt smyku), já použil faktor 0.05.

10 Land Rover

Nyní už je čas provést import připraveného modelu vozidla. Po importu je nutné nastavit správné materiály a vlastnosti konkrétního auta jako například křivka průběhu motoru, typ pohonu kol nebo vlastnosti převodoky. A zde je také čas auto rozpohybovat.

Naimportoval jsem tedy exportovaný FBX model a při importu zvolil `Import normals and tangents`. Bez změny této volby se UE snažil vytvořit normály sám, což v mém případě nefungovalo. Takto importovaný model jsem vytvořil blueprint typu `AutoSEfekty`, který už má mnoho funkcí připravených. Neslouží však k poskytování konkrétních informací o autech, proto je zde nutné nastavit křivku průběhu motoru, nastavení šestistupňové automatické převodovky a jednotlivých převodových poměrů či také změnit typ pohonu na odpovídající 4x4. Dále jsem vytvořil několik materiálů například pro barvu karoserie, hliníková kola, některé i s texturou, například budíky či zadní SPZ. Epic Games však poskytuje skvělý balíček Automotive Materials [20], který v sobě zahrnuje velice zdařilé materiály a i já některé z nich přepoužil, například pro kůži či skla.

Animační blueprint

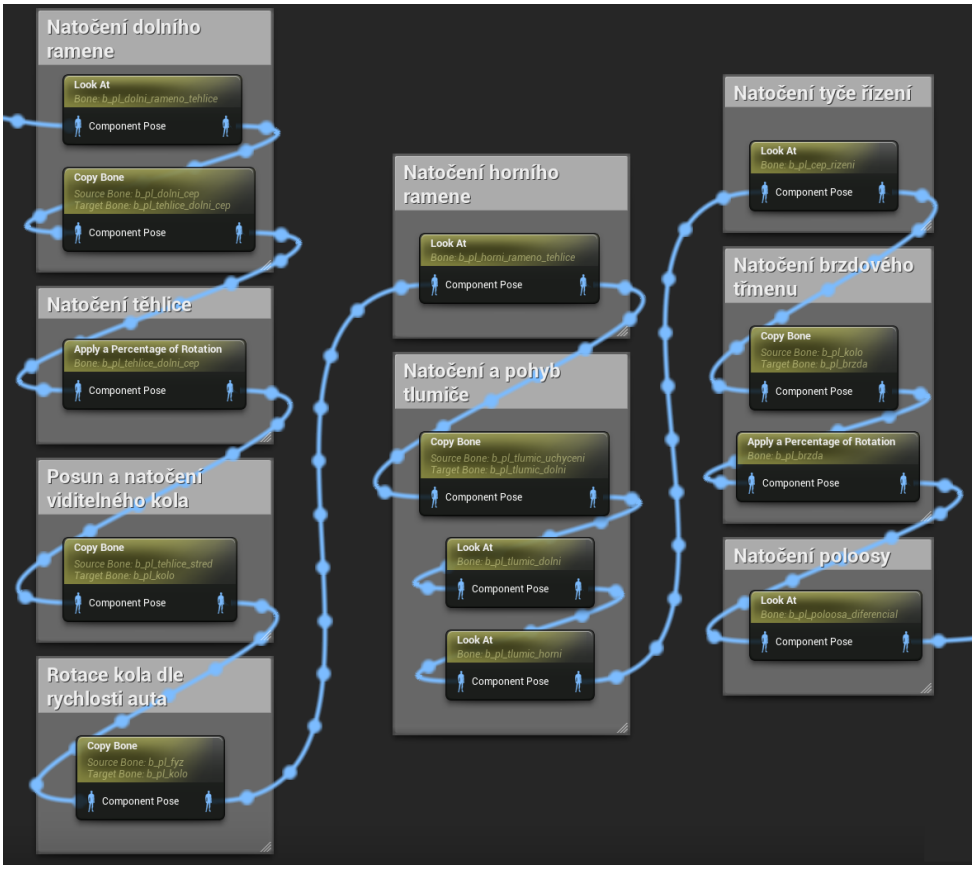
Pro naimportovaný model je nutné vytvořit `Animation Blueprint`, který bude mít na starost manipulace s kostmi, které se následně projeví transformacemi objektů na ně připojených. Při vytváření tohoto blueprintu je nutné dbát na vytvoření animačního blueprintu typu `VehicleAnimInstance` a pro správný importovaný skeleton.

Pro přehlednost jsem nutné akce vypsal do tabulky 2. Kroky 1 a 2 řeší správné natočení spodního ramene. Natočení těhlice se provádí v kroku 3. Jelikož na těhlici je umístěno kolo, je další krok posun a natočení viditelného kola. Posunuté kolo je také nutno správně rotovat dle rychlosti auta, což zaručí krok 5. V kroku 6 dojde ke správnému natočení horního ramene. Kroky 7 až 9 řeší správný pohyb tlumiče, nejprve správné nastavení spodního uchycení tlumiče a poté i směr, aby nedocházelo k deformacím. V kroku 10 dochází ke správnému natočení tyče řízení. Správná poloha a natočení brzdového tlumiče je provedena v krocích 11 a 12. A konečně poslední krok 13 řeší správnou polohu poloosy. Grafické znázornění možno vidět na obrázku 31.

V tabulce jsou všechny nutné akce pro přední levé kolo. Pro ostatní je postup velice podobný, pro pravé přední kolo je nutné prohození znamének v krocích 10 a 13. Jelikož zadní náprava se nenatáčí, není zde ani žádná tyč řízení, krok 10 u ní tedy nedává smysl. Rovněž se může i přeskočit natočení brzdového třmene v kroku 12.

	Akce	**Zdrojová kost (pro Look At modifikovaná)**	**Cílová kost (pro Look At cíl pohledu)**	**Nastavené parametry**
1	`Look At`	`b_pl_dolni_rameno_tehlice`	`b_pl_fyz`	Sledovat pouze v ose X (1, 0, 0)
2	`Copy Bone`	`b_pl_dolni_cep`	`b_pl_tehlice_dolni_cep`	Pouze translace
3	`Apply a Percentage of Rotation`	`b_pl_tehlice_dolni_cep`	`b_pl_fyz`	Pouze osu Z
4	`Copy Bone`	`b_pl_tehlice_stred`	`b_pl_kolo`	Pouze translace a rotace
5	`Copy Bone`	`b_pl_fyz`	`b_pl_kolo`	Pouze rotace
6	`Look At`	`b_pl_horni_rameno_tehlice`	`b_pl_horni_cep`	Sledovat pouze v ose X (1, 0, 0)
7	`Copy Bone`	`b_pl_tlumic_uchyceni`	`b_pl_tlumic_dolni`	Pouze translace
8	`Look At`	`b_pl_tlumic_dolni`	`b_pl_tlumic_horni`	Sledovat pouze v ose Z (0, 0, 1)
9	`Look At`	`b_pl_tlumic_horni`	`b_pl_tlumic_dolni`	Sledovat pouze v záporné ose Z (0, 0, -1)
10	`Look At`	`b_pl_cep_rizeni`	`b_pl_servorizeni`	Sledovat pouze v ose Y (0, 1, 0)
11	`Copy Bone`	`b_pl_kolo`	`b_pl_brzda`	Pouze translace
12	`Apply a Percentage of Rotation`	`b_pl_brzda`	`b_pl_kolo`	Pouze osu Z
13	`Look At`	`b_pl_poloosa_diferencial`	`b_pl_kolo`	Sledovat pouze v záporné ose Y (0, -1, 0)

Tabulka 2: Tabulka animací pro přední levé kolo

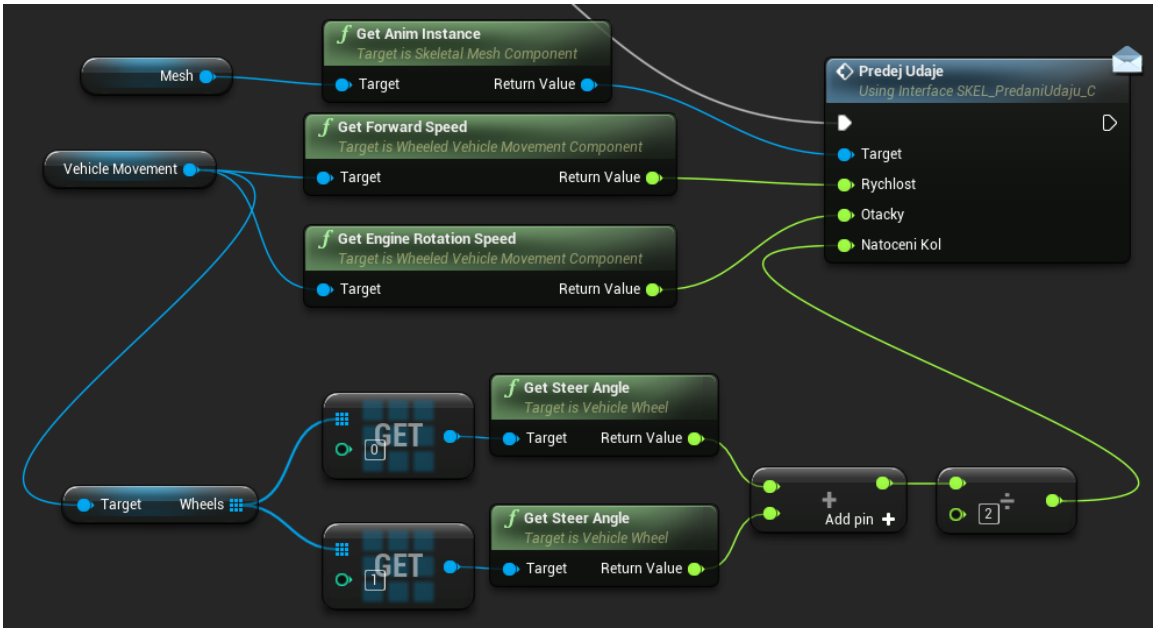


Obrázek 31: Kompletní animace pro přední levé kolo

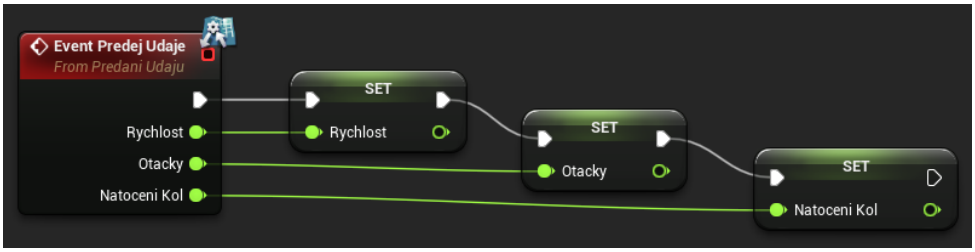
Animace ukazatelů budíků a volantu

Animační blueprint nemá standardně přístup k informacím o pohybu auta, jako je jeho rychlost, otáčky motoru či natočení kol. Řešení jsem objevil v pohodě rozhraní (`Blueprint Interface`). Vytvořil jsem instanci právě tohoto typu s názvem `PredaniUdaju` s jednou funkcí `PredejUdaje`. Ta má tři vstupy, přesně jako počet údajů, které potřebuji, pro rychlost, otáčky a natočení kol. V `Event Graph` animačního blueprintu jsem si rovněž vytvořil 3 proměnné, které nastavím při obsloužení události `PredejUdaje`. Dále je nutné také nastavit animačnímu blueprintu, aby o tomto rozhraní věděl. To se dá provést v nastavení `ClassSettings`, přičemž se zobrazí vlastnosti třídy s možností `Implemented Interfaces`, kde stačí toto rozhraní přidat.

V rámci blueprintu `AutoSEfekty`, kde je přístup k informacím o pohybu auta, pak tuto metodu `PredejUdaje` volám s nastavením požadovaných hodnot, což můžete vidět na obrázku 32. Obrázek 33 zase ukazuje obloužení této události v `EventGraph` v rámci animačního blueprintu a uloží hodnoty do proměnných. Tyto hodnoty jsou pak už známy `AnimGraph`, kde po vhodné úpravě provedu samotnou rotaci objektů, viz obrázek 34.



Obrázek 32: Vyvolání události předání údajů v blueprintu AutoSEfekty



Obrázek 33: Obsloužení události předání údajů v animačním blueprintu

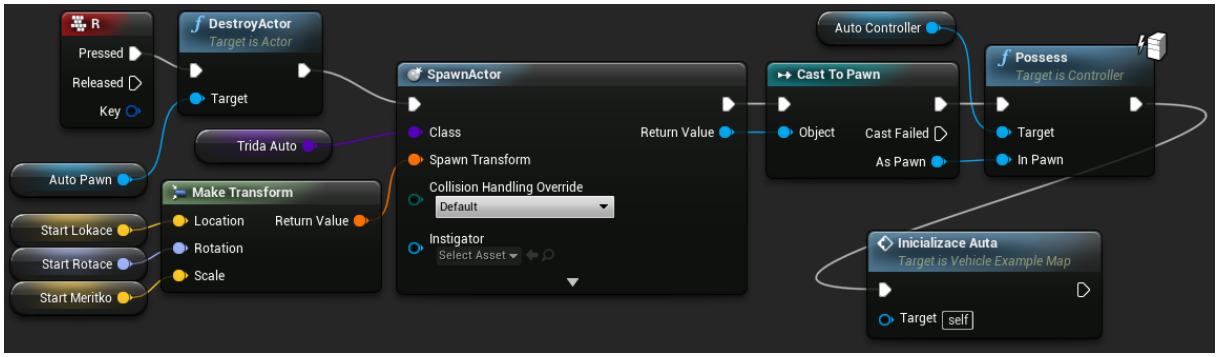
Obrázek 34: Úprava vstupů a provedení transformací v animačním blueprintu

11 Nastavení Level blueprint

Když je připraven terén a auto je pojízdné, rozhodl jsem se pro lepší ukázku funkce auta vytvořit tzv. demo režim, kdy auto objíždí předem připravenou trasu. Do terénu jsem rozmístil několik `TriggerBox`ů, které mi určují trasu auta. Přidal jsem je do připravené proměnné typu pole. Aby mohl uživatel sledovat, ke kterému bodu trasy má auto právě namířeno, vytvořím automaticky při inicializaci simulace nad jednotlivými body aktéra typu `Text Render`, kterému jako text nastavím číselné označení podle toho, kolikátý je tento cíl v pořadí (a tedy také v poli). Aby nebyl o aktuálním cíli pochyb, je vždy popisek obarven červenou barvou, přičemž ostatní popisky mají barvu bílou. A jelikož nehybné popisky by nebyly v určitých úhlech vidět, přidal jsem také jejich rotaci při každé události `Tick`. Zároveň může uživatel aktuální cíle měnit, stačí použít klávesy `-` a `+` na numerické klávesnici, po jejichž stisku se index cílového bodu buď sníží nebo zvýší. Samotný režim demo jízdy se spustí klávesou `.` rovněž na numerické klávesnici. Při demo režimu má uživatel možnost nastavit rychlost pomocí kláves `8` a `5`.

V rámci level blueprintu je také definováno přepínání režimu automatické jízdy, kdy má uživatel možnost zvolit rychlost či natočení kol a auto toto nastavení dodržuje, pokud není změněno. Tento režim se zapíná tlačítkem `0`, opět na numerické klávesnici. O zapnutí či vypnutí těchto režimů dostane člověk textovou zprávu, stejně tak o změně požadované rychlosti či natočení kol.

Při jízdě auta se může stát, že vozidlo se zasekne na podvozku nebo třeba skončí na boku či střeše. Aby nebylo nutné simulaci zapínat znova, přidal jsem možnost návratu na startovní pozici stiskem tlačítka `R`. Při jednoduché teleportaci si však vozidlo nadále drželo aktuální fyzikální vlastnosti (rychlost, směr pohybu) a docházelo k nepředvídatelným akcím, proto jsem to vyřešil jinak. Nedojde čistě k přesunu, nýbrž ke zrušení starého objektu auta a jeho novou inicializací. Údaje nutné k úspěšnému provedení restartu (jako například startovní pozice, rotace, měřítko nebo také blueprint třída aktuálního vozidla) jsou automaticky uloženy po zapnutí simulace mou funkcí `Inicializace Auta`. Tato funkce se zároveň volá i při úspěšném restartování pro správnou inicializaci proměnných. Proces znázorňuje obrázek 35.



Obrázek 35: Restartování simulace

12 Závěr

Diplomová práce mi umožnila seznámení a hlavně propojení mnoha rozličných odvětví, nejen z mého oboru, v jeden celek. Nejprve jsem musel nastudovat lichoběžníkovou nápravu a pochopit její princip a fungování. Poté se mi povedlo sehnat bezplatný model vozidla Range Rover Sport HSE z roku 2010. Bezplatné modely, podle mých zkušeností, mají občas několik, místy zásadních, chyb. Nejinak tomu bylo i v mém případě. Musel jsem model opravit, zejména normály ploch a kompletně znovu nastavit materiály, neboť valná většina byla duplicitních. Poté už jsem se mohl pustit do vymodelování náprav a vzápětí na to i vytvoření a rozmístění kostí, které jednotlivé součásti spojují. Poslední krok byl nutný v součinosti s prostředím Unreal Engine, neboť jsem musel zjistit, jak model připravit právě pro použití v tomto enginu.

V rámci Unreal Engine jsem vytvořil komplexní terén s překážkami, na kterém následně demonstruji funkci nápravy. Aby byla simulace co nejvěrohodnější, zanechává auto na určitých površích odpovídající stopy po pneumatikách a také prach či létající kamínky od pneumatik a jiné. Pro generování stop jsem zvolil použití blueprintů a materiálu typu Deferred Decal. Parametry jako velikost či rotaci stop nastavuji podle aktuálních situace jako je rychlost či směr natočení kola a stejně tak využívám materiál, který taktéž dynamicky nastavuji. Nachází-li se auto ve smyku, je opět vhodným nastavením stop a jejich materiálu vytvořen odpovídající efekt viditelný jako směs čar ve směru pohybu vozidla dle nastaveného vzorku pneumatik. Správu efektů jsem pro změnu řešil vlastním C++ kódem, který rozšiřuje standardní třídu pro auto. Tato třída řeší správné přepínání efektů dle typu povrchu. Pro snadné použití požadovaných efektů jsem také vytvořil pomocnou datovou třídu, kde pro každý typ povrchu nastavím požadovaný částicový systém a efekt stop. Zároveň jsou zde také pravdivostní proměnné, zda-li se mají efekty zobrazovat pouze při smyku (asfalt, skála) nebo vždy (tráva, šotolina). Snímky obrazovky z výsledné simulace jsou k nalezení v příloze A.

Při práci jsem se snažil dodržovat vhodnou dědičnost tříd a rozmístění třídních metod, aby rozšíření projektu o další automobily bylo co nejrychlejší a nemusela se obecná funkcionalita implementovat znovu. Naimplementoval jsem také jízdu podle zadané rychlosti s natočením volantu, jenž drží nastavenou polohu. Také je možno zapnout demo režim, kdy vozidlo projíždí předem nastavenou trasu, která se opět dá jednoduše měnit. Pokud při simulace dojde k potížím, kdy například vozidlo uvázne, je možno auto stiskem tlačítka simulaci restartovat do výchozího nastavení.

V rámci školních předmětů jsem se již setkal s modelováním, nikdy se však nejednalo o pokročilejší použití v jiných programech. Systém kostí jsem také dosud neznal a jeho použití na reálném projektu mi dalo náhled, jak užitečné ale zároveň taky pracné mohou být. Zároveň jsem nakoukl pod pokličku možností MaxScriptu, který je součástí modelovacího nástroje 3ds max a díky kterému si může uživatel vytvořit vlastní skripty či funkce, které dokáží práci velice usnadnit.

S Unreal Enginem jsem se v minulosti setkal pouze se statickými modely, takže i zde jsem

se dozvěděl mnoho nových informací. Zejména jsem dostal možnost porovnat blueprinty a programování v C++. Ukázalo se, že blueprinty nejsou jen jakési šidítko, ale plnohodnotný nástroj, velice podobný klasickému programování, avšak v modernějším kabátku. I při použití blueprintů je nutné mít v hlavě ten správný algoritmus, jehož chcete dosáhnout. Jeho implementace však byla v mém případě o mnoho rychlejší v blueprintech než v C++. Zároveň jsem se ale také přesvědčil, že programování i v takto rozsáhlém enginu je pro běžného smrtelníka stravitelné, zásluhu na tom také má dobrá dokumentace a aktivní komunita lidí, kteří engine používají.

Reference

[1] The History Of Racing Games *IGN* [online]. © 2015 [cit. 2018-04-16]. Dostupné z: http://uk-microsites.ign.com/the-history-of-racing-games/

[2] Game Engine Technology by Unreal *Unreal Engine* [online]. © 2018 [cit. 2018-04-16]. Dostupné z: http://www.unrealengine.com

[3] Unity *Unity* [online]. © 2018 [cit. 2018-04-16]. Dostupné z: https://unity3d.com/

[4] An Open-Source Physics Engine *Project Chrono* [online]. © 2018 [cit. 2018-04-16]. Dostupné z: http://projectchrono.org/

[5] V-REP: Create. Compose. Simulate. Any Robot. *Coppelia Robotics* [online]. © 2018 [cit. 2018-04-16]. Dostupné z: http://www.coppeliarobotics.com/

[6] VLK, František. Podvozky motorových vozidel. 3., přeprac., rozš. a aktualiz. vyd. Brno: František Vlk, 2006. ISBN 80-239-6464-x.

[7] JAN, Zdeněk, Bronislav ŽDÁNSKÝ a Jiří ČUPERA. Automobily. 2., aktualiz. vyd. Brno: Avid, 2009. ISBN 978-80-87143-11-7.

[8] Control Arms *Delphi* [online]. 2018 [cit. 2018-04-10]. Dostupné z: https://www.delphiautoparts.com/en-US/parts/steering-suspension/control-arms

[9] Nápravy *OHV Racing* [online]. 2018 [cit. 2018-04-10]. Dostupné z: http://ohvracing.cz/napravy/

[10] PV, Satheesh. Unreal Engine 4 Game Development Essentials: Master the basics of Unreal Engine 4 to build stunning video games. Birmingham: Packt Publishing, 2016. ISBN 978-1-78439-196-6.

[11] SEWELL, Brenden. Blueprints Visual Scripting for Unreal Engine: Build professional 3D games with Unreal Engine 4's Visual Scripting system. Birmingham: Packt Publishing, 2015. ISBN 978-1-78528-601-8.

[12] KADLEC, Václav. Učíme se programovat v jazyce C. Praha: Computer Press, 2002. ISBN 9788072267156.

[13] PRATA, Stephen. Mistrovství v C++. 3., aktualiz. vyd. Přeložil Boris SOKOL. Brno: Computer Press, 2007. Bestseller (Computer Press). ISBN 978-80-251-1749-1.

[14] 3D model Land Rover Range Rover Sport HSE 2010 *CGTrader* [online]. © 2017 [cit. 2017-10-30]. Dostupné z: https://www.cgtrader.com/free-3d-models/car/suv/land-rover-range-rover-sport-hse-2010

[15] Land Rover. (2010). Range Rover Sport: Owner Manual.

[16] Learn & Explore 3ds max *Autodesk.com* [online]. © 2017 [cit. 2017-11-8]. Dostupné z: https://knowledge.autodesk.com/support/3ds-max/learn-explore

[17] Textures for 3D, graphic design and Photoshop! *textures.com* [online]. © 2017 [cit. 2017-04-12]. Dostupné z: https://www.textures.com/

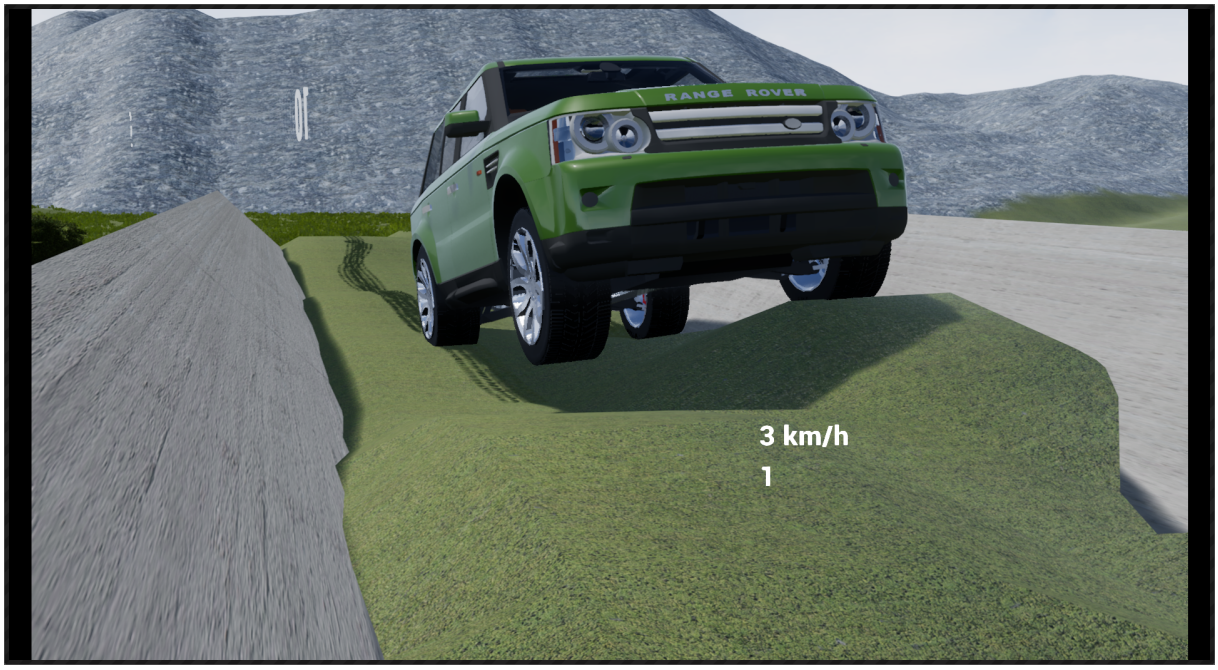
[18] NormalMap-Online *Github cpetry* [online]. © 2017 [cit. 2017-04-12]. Dostupné z: http://cpetry.github.io/NormalMap-Online/

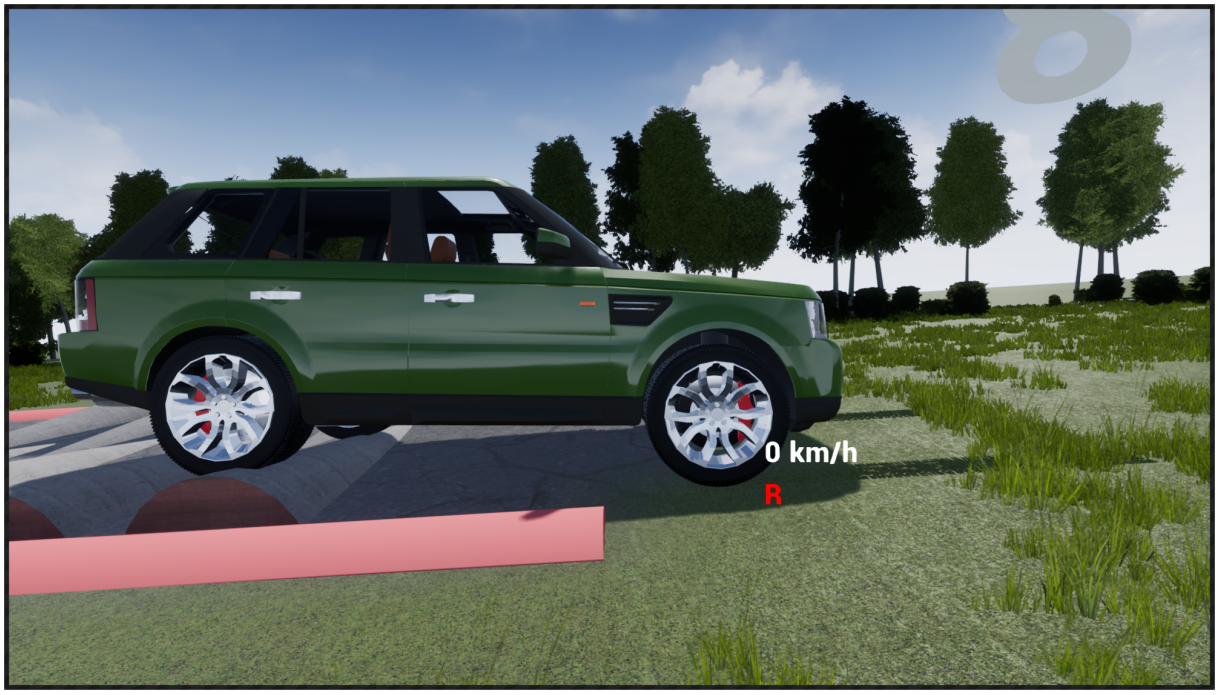
[19] Free Foliage Starter Kit *Unreal Engine Forums* [online]. © 2017 [cit. 2017-04-12]. Dostupné z: https://forums.unrealengine.com/community/community-content-tools-and-tutorials/30694-free-foliage-starter-kit

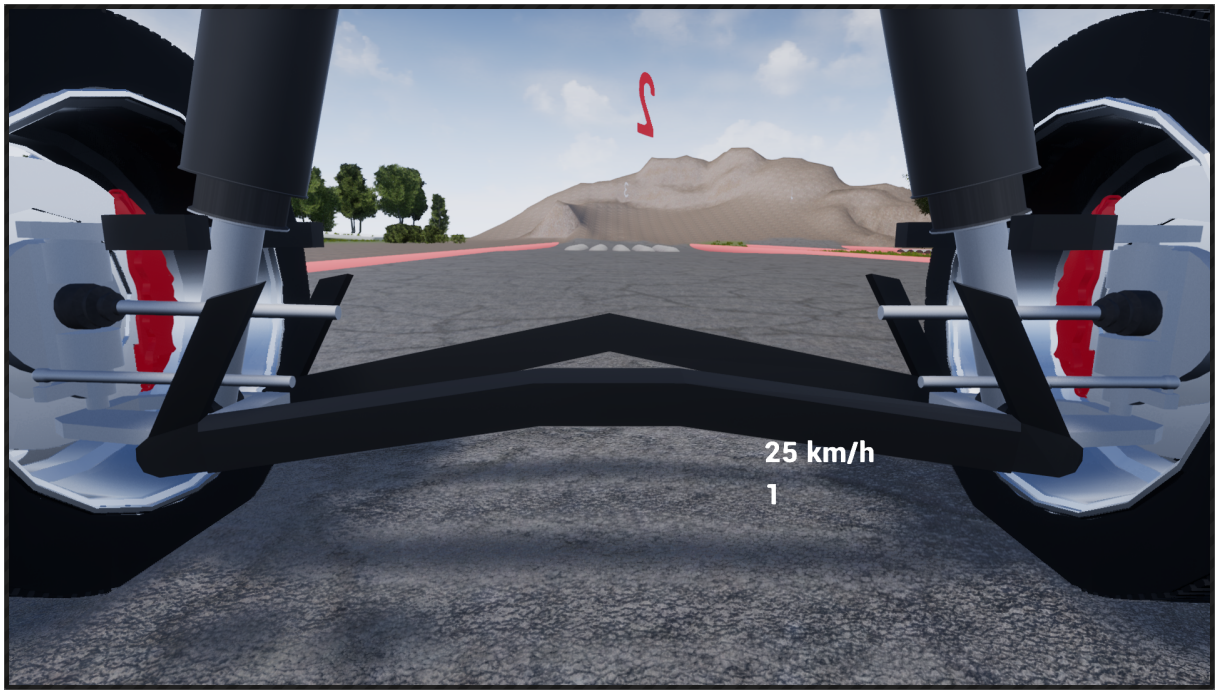
[20] Automotive Materials *Unreal Engine Marketplace* [online]. © 2017 [cit. 2017-04-12]. Dostupné z: https://www.unrealengine.com/marketplace/automotive-material-pack

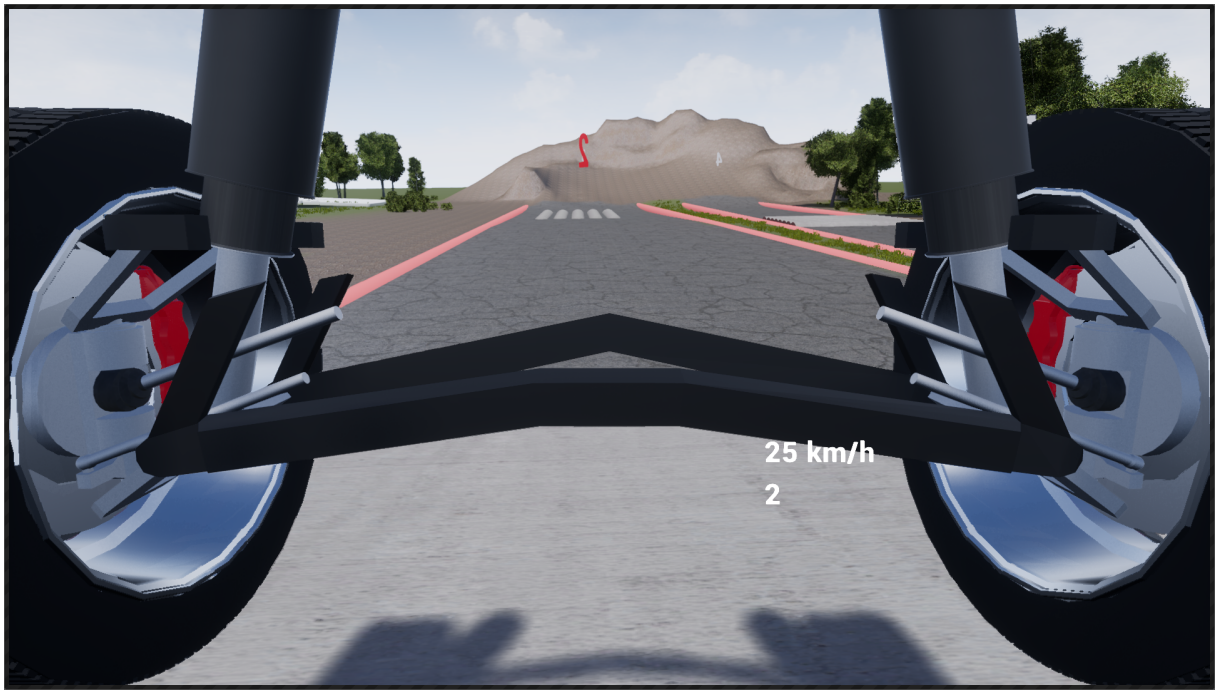
[21] 6HP28, 6 speed automatic transmission for cars *ZF* [online]. © 2011 [cit. 2018-04-16]. Dostupné z: https://www.zf.com/global/media/product_media/cars_5/cars_driveline_6_speed_automatic_transmission/pdf_140/6HP28_DataSheet.pdf

A Snímky obrazovky ze simulace

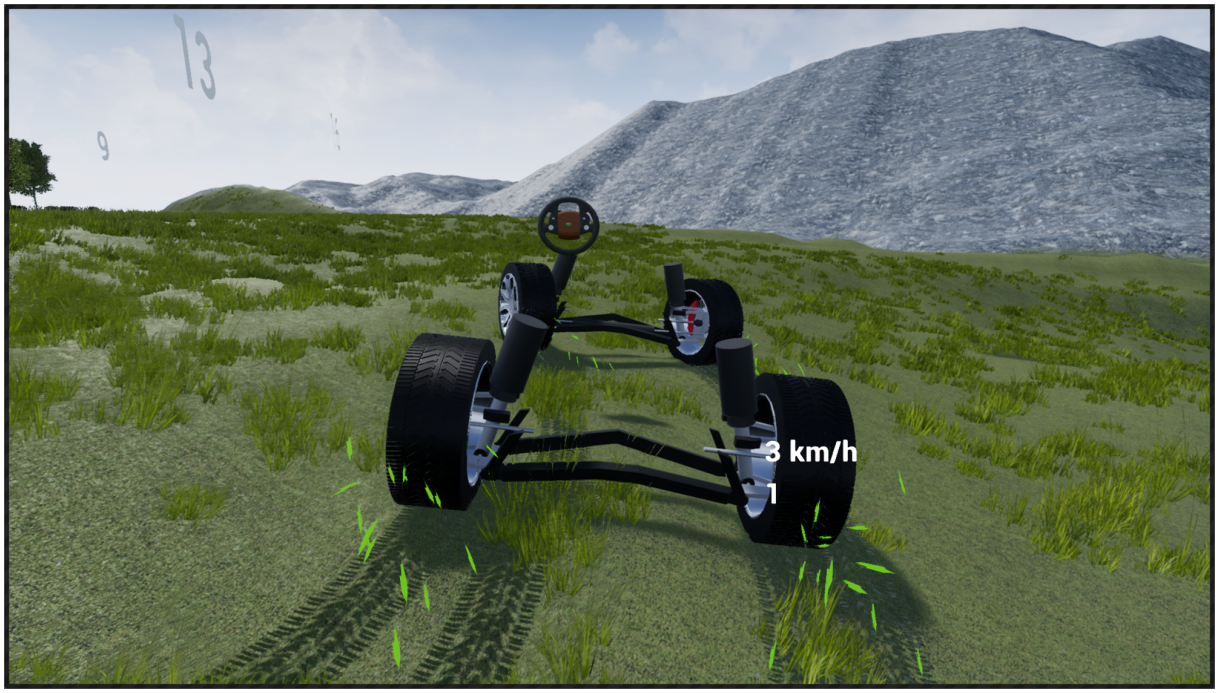

RANGE ROVER
3 km/h
1


0 km/h
R


25 km/h
1


25 km/h
2


HAJ OVSKY
0 km/h
2


3 km/h
1


28 km/h
1
LAND ROVER

B MaxScript funkce

```
fn plv obj =
(
    verts = obj.selectedVerts
    p = Point3 0 0 0
    for v~in verts do
       p += v.pos
    p = p/verts.count
    return p
)

fn plvs obj =
(
   obj.pivot = plv(obj)
)

fn plf obj =
(
   faces = polyop.getFaceSelection obj
   p = Point3 0 0 0
   num = 0
   for face in faces do
   (
      p += polyop.getFaceCenter obj face
      num += 1
   )
   p /= num
   return p
)

fn plfs obj =
(
   obj.pivot = plf(obj)
)
```

Výpis 2: Vlastní MaxScript funkce

C Hlavičkový soubor třídy NastaveniEfektu

```
#pragma once

#include "CoreMinimal.h"
#include "Engine/DataAsset.h"
#include "Particles/ParticleSystem.h"
#include "PhysicalMaterials/PhysicalMaterial.h"
#include "NastaveniEfektu.generated.h"

#define VEHICLE_SURFACE_Default  SurfaceType_Default
#define VEHICLE_SURFACE_Asfalt  SurfaceType1
#define VEHICLE_SURFACE_Sotolina SurfaceType2
#define VEHICLE_SURFACE_Trava  SurfaceType3
#define VEHICLE_SURFACE_Skala  SurfaceType4

UCLASS()
class HAJ0094_API UNastaveniEfektu : public UDataAsset
{
    GENERATED_BODY()
public:
    // asfalt
    UPROPERTY(EditDefaultsOnly, Category = Efekty)
        UParticleSystem* Asfalt;

    UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadWrite)
        bool AsfaltEfektPouzePriSmyku;

    UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadWrite)
        UMaterial* AsfaltStopy;

    UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadWrite)
        bool AsfaltStopyPouzePriSmyku;

    // sotolina
    UPROPERTY(EditDefaultsOnly, Category = Efekty)
        UParticleSystem* Sotolina;

    UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadWrite)
        bool SotolinaEfektPouzePriSmyku;

    UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadWrite)
        UMaterial* SotolinaStopy;

    UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadWrite)
        bool SotolinaStopyPouzePriSmyku;

    // trava
    UPROPERTY(EditDefaultsOnly, Category = Efekty)
        UParticleSystem* Trava;

    UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadWrite)
        bool TravaEfektPouzePriSmyku;

    UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadWrite)
        UMaterial* TravaStopy;

    UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadWrite)
        bool TravaStopyPouzePriSmyku;

    // skala
    UPROPERTY(EditDefaultsOnly, Category = Efekty)
        UParticleSystem* Skala;

    UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadWrite)
        bool SkalaEfektPouzePriSmyku;

    UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadWrite)
        UMaterial* SkalaStopy;

    UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadWrite)
        bool SkalaStopyPouzePriSmyku;
```

```
public:
    UParticleSystem * ZvolitEfekt(UPhysicalMaterial* PhysMaterial, float CurrentSpeed);
    bool JeAktivni(UPhysicalMaterial* PhysMaterial, bool slide);
};
```

Výpis 3: NastaveniEfektu.h

D Zdrojový soubor třídy NastaveniEfektu

```
#include "NastaveniEfektu.h"
#include "Engine/Engine.h"

UParticleSystem* UNastaveniEfektu::ZvolitEfekt(UPhysicalMaterial* PhysMaterial, float CurrentSpeed)
{
    EPhysicalSurface SurfaceType = UPhysicalMaterial::DetermineSurfaceType(PhysMaterial);
    if (SurfaceType == VEHICLE_SURFACE_Asfalt)
        return Asfalt;
    else if (SurfaceType == VEHICLE_SURFACE_Sotolina)
        return Sotolina;
    else if (SurfaceType == VEHICLE_SURFACE_Trava)
        return Trava;
    else if (SurfaceType == VEHICLE_SURFACE_Skala)
        return Skala;
    else
        return Asfalt;
}

bool UNastaveniEfektu::JeAktivni(UPhysicalMaterial* PhysMaterial, bool slide)
{
    EPhysicalSurface SurfaceType = UPhysicalMaterial::DetermineSurfaceType(PhysMaterial);
    if (slide)
        return true;
    if (SurfaceType == VEHICLE_SURFACE_Asfalt)
        return !AsfaltEfektPouzePriSmyku;
    else if (SurfaceType == VEHICLE_SURFACE_Sotolina)
        return !SotolinaEfektPouzePriSmyku;
    else if (SurfaceType == VEHICLE_SURFACE_Trava)
        return !TravaEfektPouzePriSmyku;
    else if (SurfaceType == VEHICLE_SURFACE_Skala)
        return !SkalaEfektPouzePriSmyku;
    else
        return !AsfaltEfektPouzePriSmyku;
}
```

Výpis 4: NastaveniEfektu.cpp

E Hlavičkový soubor třídy HAJ0094_efekty

```
#pragma once

#include "CoreMinimal.h"
#include "HAJ0094Pawn.h"
#include "NastaveniEfektu.h"
#include "WheeledVehicle.h"
#include "Particles/ParticleSystemComponent.h"
#include "HAJ0094_efekty.generated.h"

UCLASS()
class HAJ0094_API AHAJ0094_efekty : public AHAJ0094Pawn
{
   GENERATED_BODY()

protected:
   UPROPERTY(Category = Efekty, EditDefaultsOnly, BlueprintReadOnly)
      UNastaveniEfektu* NastaveniEfektu;

   UPROPERTY(Transient)
      UParticleSystemComponent* EfektKolo[4];

public:
   float VratRychlostAuta() const;
   void InicializaceEfektu(int IndexKola);
   void AktualizaceEfektu();
   virtual void Tick(float Delta) override;
};
```

Výpis 5: HAJ0094_efekty.h

F Zdrojový soubor třídy HAJ0094_efekty

```
#include "HAJ0094_efekty.h"
#include "Components/SkeletalMeshComponent.h"
#include "WheeledVehicleMovementComponent.h"

float AHAJ0094_efekty::VratRychlostAuta() const
{
    return FMath::Abs(GetVehicleMovement()->GetForwardSpeed()) * 0.036f;
}

void AHAJ0094_efekty::InicializaceEfektu(int IndexKola)
{
    // inicializace cisteho efektu
    EfektKolo[IndexKola] = NewObject<UParticleSystemComponent>(this);
    EfektKolo[IndexKola]->bAutoActivate = false;
    EfektKolo[IndexKola]->bAutoDestroy = false;
    EfektKolo[IndexKola]->RegisterComponentWithWorld(GetWorld());
    // z nazvu kosti fyzickeho kola pr. b_pl_fyz
    // vezmi prvnich 5 znaku a pripoj tento suffix
    FString suffix = "kolo_efekty";
    // b_pl_fyz --> b_pl_kolo_efekty
    FString bone_efekty_name = GetVehicleMovement()->WheelSetups[IndexKola].BoneName.ToString().Left(5).Append(suffix);
    // pripojeni efektu k autu a nastaveni jeho lokace ke spravne kosti, s tim, ze zdroj efektu bude nasledovat spravnou pozici
    EfektKolo[IndexKola]->AttachToComponent(GetMesh(), FAttachmentTransformRules::KeepRelativeTransform, FName(*
        bone_efekty_name));
}

void AHAJ0094_efekty::AktualizaceEfektu()
{
    if (NastaveniEfektu && GetVehicleMovement() && GetVehicleMovement()->Wheels.Num() > 0)
    {
        const float AktualniRychlost = VratRychlostAuta();
        // aktualizace efektu pro kazde kolo
        for (int32 IndexKola = 0; IndexKola < ARRAY_COUNT(EfektKolo); IndexKola++)
        {
            // je-li rychlost auta prilis pomala, vypni efekty
            if (AktualniRychlost < 5 && EfektKolo[IndexKola] != nullptr)
            {
                EfektKolo[IndexKola]->SetActive(false);
                continue;
            }
            // zjisteni, na jakem povrchu se auto pohybuje
            UPhysicalMaterial* MaterialDotyku = GetVehicleMovement()->Wheels[IndexKola]->GetContactSurfaceMaterial();
            // pokud se nachazi kolo ve vzduchu, vypni efekty
            if (MaterialDotyku == nullptr && EfektKolo[IndexKola] != nullptr)
                EfektKolo[IndexKola]->SetActive(false);
            // zjisteni pozadovaneho efektu v zavislosti na materialu
            UParticleSystem* PozadovanyEfekt = NastaveniEfektu->ZvolitEfekt(MaterialDotyku, AktualniRychlost);
            // zjisteni, zda-li je nejaky efekt aktualne zapnut
            const bool JeAktivni = EfektKolo[IndexKola] != nullptr && !EfektKolo[IndexKola]->bWasDeactivated && !EfektKolo[IndexKola
                ]->bWasCompleted;
            // zjisteni aktualniho efektu
            UParticleSystem* AktualniEfekt = EfektKolo[IndexKola] != nullptr ? EfektKolo[IndexKola]->Template : nullptr;
            // pokud je pozadovany efekt jiny nez aktualni
            if (PozadovanyEfekt != nullptr && (AktualniEfekt != PozadovanyEfekt || !JeAktivni))
            {
                // pokud je efekt aktivni, vypni jej
                if (EfektKolo[IndexKola] == nullptr || !EfektKolo[IndexKola]->bWasDeactivated)
                {
                    // pokud ma kolo nejaky efekt, zrus jej
                    if (EfektKolo[IndexKola] != nullptr)
                    {
                        EfektKolo[IndexKola]->SetActive(false);
                        EfektKolo[IndexKola]->bAutoDestroy = true;
                    }
                    // inicializace cisteho efektu
                    InicializaceEfektu (IndexKola);
                }
                // nastaveni efektu na pozadovany a jeho zapnuti
                EfektKolo[IndexKola]->SetTemplate(PozadovanyEfekt);
                EfektKolo[IndexKola]->ActivateSystem();
            }
```

```
            // pokud nema byt zobrazen zadny efekt , vypni stavajici
            else if (PozadovanyEfekt == nullptr && JeAktivni)
            {
                EfektKolo[IndexKola]->SetActive(false);
            }
            //  zjisteni , zda se auto nachazi ve smyku
            bool Smyk = GetVehicleMovement()->CheckSlipThreshold(0.5, 0.5);
            //  zjisteni , zda-li ma byt efekt aktivovan a jeho aktivace
            bool Zapnout = NastaveniEfektu->JeAktivni(MaterialDotyku, Smyk);
            EfektKolo[IndexKola]->SetActive(Zapnout);
        }
    }

}

void AHAJ0094_efekty::Tick(float Delta)
{
    Super::Tick(Delta);
    AktualizaceEfektu();
}
```

Výpis 6: HAJ0094_efekty.cpp